University of Alberta

Library Release Form

Name of Author: David Ryan Erickson

Title of Thesis: Non-Learning Artificial Neural Network Approach to Real-Time Motion Planning for the Pioneer Robot

Degree: Master of Science

Year this Degree Granted: 2003

University of Alberta

NON-LEARNING ARTIFICIAL NEURAL NETWORK APPROACH TO REAL-TIME MOTION PLANNING FOR THE PIONEER ROBOT

by

David Ryan Erickson

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Master of Science.

Department of Electrical and Computer Engineering

Edmonton, Alberta

Spring 2003

University of Alberta

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **Non-Learning Artificial Neural Network Approach to Real-Time Motion Planning for the Pioneer Robot** submitted by David Ryan Erickson in partial fulfillment of the requirements for the degree of **Master of Science**.

Dedication

To my wife Michelle,

for believing that this was possible.

No man is free who is not master of himself -Epictus

Abstract

The real-time motion planning of mobile robotics is a stumbling block for the expansion of mobile robotics into more complex fields. The solution of this problem is the focus of this thesis. This thesis builds upon the work of Dr. Simon Yang and Prof. Max Meng, which implements a biologically-inspired non-learning artificial neural network (ANN) This ANN computes a motion path for the Pioneer 2 DX mobile robot under the Saphira operating system. It also independently confirms the findings in the earlier work. This method is a variation on the approximate cell decomposition method where neurons represent free space or regions occupied by obstacles. Each neuron in the neural network is characterized by an additive or shunting equation that models the interaction of obstacle neurons, free space neurons and the neuron representing the target pose. This method is able to find a path if one exists from any arbitrary initial and final pose.

Acknowledgements

I wish to thank my supervisor, for giving me enough freedom to explore my topic, and for his advice when I needed it. He gave me enough rope to find my way, and when you are learning to explore and confirm the research for yourself, this is a big help. His advice and assistance were always ready when called upon. I would like to also like to thank Dr. Simon Yang who helped me when I bogged down with the details of implementation. I tried not to rely on his text directly and this led me to linger more than necessary on the implementation.

I would also like to thank my wife and children. Their joy of life and happiness gave me peace of mind to direct my attention elsewhere for a period of time. I know I have promised them a new and better future. With this document's completion I can set about completing this promise.

Table of Contents

List of Tables

List of Figures

List of Symbols

Symbols Used in Robot Motion Planning

$\{0\}$	Global frame of reference for the workspace W
$\{R\}$	Robot frame of reference fixed on the robot origin R
Δt	unified time step for finite difference approximation methods
α	Runge-Kutta time step coefficient
β	Runge-Kutta time step coefficient
θ	Orientation angle of robot $\{R\}$ with respect to $\{0\}$
τ	Path of robot
υ	Translational velocity vector
ω	Neural weighting factor
ξ	Activation level of neuron
A	Decay rate
b	Diameter of drive axis
B	Upper bound of neural activation
C_1	2^{nd} order Runge-Kutta equation
C_2	2^{nd} order Runge-Kutta equation
C	Configuration space
C_m	Membrane capacitance
CT	Time-varying configuration space
d	Distance metric
D	Lower bound of neural activity
E_P	Nemst potential of passive channels in neuron
E_K	Nemst potential of potassium
E_{Na}	Nemst potential of sodium
F_1	3^{rd} order Bogacki-Shampine equation
F_2	3^{rd} order Bogacki-Shampine equation
F_3	3^{rd} order Bogacki-Shampine equation
F_4	3^{rd} order Bogacki-Shampine equation
F	Region of configuration q
$\Im$	Set of all free space in workspace

g	Neural input gain factor
g_K	Conductance of potassium
g_{Na}	Conductance of sodium
g_P	Conductance of passive channels
$[I_i]^+$	Target input to neuron i
$[I_i]^-$	Obstacle input to neuron i
K_1	4^{th} order Runge-Kutta equation
K_2	4^{th} order Runge-Kutta equation
K_3	4^{th} order Runge-Kutta equation
K_4	4^{th} order Runge-Kutta equation
K(q)	State space model of Pioneer
N	Neural region dimension
NP	Non-polynomial computational complexity
O_i	Obstacle i within the workspace
p_c	Current position in path search
p_n	Next position in path search
p_t	Target position in path search
P	Polynomial computational complexity
PSPACE	Polynomial space computational complexity
q	configuration of robot R
R	Robot within workspace
R^n	Real space of n dimension
$s_i^+(t)$	Positive input to Grossberg equation
$s_i^-(t)$	Positive input to Grossberg equation
v_l	Left drive wheel velocity
v_r	Right drive wheel velocity
V_m	Voltage across neural membrane
$V(\xi)$	Lyapunov function
W	Robot Workspace, universe of discourse for robot motion planning
x	Cartesian plane x-axis
y	Cartesian plane y-axis

Chapter 1

Introduction

This thesis describes the implementation of an artificial neural network (ANN) for motion planning on the Pioneer robot. This ANN seeks to solve the advanced motion extension to the basic motion problem, as defined by Latombe 1991, for the Pioneer robot and thereby allow motion under kinematic constraint, location uncertainty, and multiple moving objects. The advanced motion problem will be defined further on in Chapter 2. We describe the background and implementation of the ANN as well as the Pioneer robot and its operating system called Saphira, which acts as the interface between the robot and the motion planner. This thesis also describes the modifications to the Saphira robot control architecture that were implemented in order to operate the ANN within it.

In the field of mobile robot vehicles, there are three tasks that must be achieved in real-time and with a high probability of success despite an uncertain environment. These tasks are data fusion, localization, and robot motion planning. Data fusion refers to the process of collection and collation of data from either co-located sensors on the robot, or from sensors in fixed positions within the robot's environment. Data can

come from a variety of sources like a global positioning system that returns the approximate location on the surface of the earth, to a simple array of ultrasonic sensors, which the Pioneer employs, that return the distance to an obstacle in front of the robot. This data becomes a model of the environment surrounding the robot and is the basis for all other higher order functions of the robot.

Localization is the procedure of processing data to determine, within an error margin, the precise pose (position and orientation) of the robot within the environment. This is developed from various components of the data and returns a point-to-point estimate of the parameters or degrees of freedom of the robot. If further planning is to be successful, the accuracy of this data is of paramount importance.

Finally, and subsequent to the previous two elements, the robot must construct and execute a path plan to move the robot from its current state to the desired end state in order to fulfil the requirements of its application. Robot motion planning is not a simple iterative process but must constantly update the solution in keeping with the changing state of the environment based on the uncertain portion that is extracted from the sensors.

These three processes, running sequentially, are concurrent to the real time motion of the robot. One of the most important problems in the design of mobile robotics is to construct a system for motion planning that can respond immediately to changing conditions in the environment. The benefit of a planning system that can respond better than current methods is an increase in the range of applications that mobile robotics can be employed in, as well as making them more robust in the current applications in which they are currently employed.

1.1 Related Work

The background literature for all related aspects to this thesis will be described in the remainder of this chapter. Motion planning will be set out and the ANN method will be put in context with other methods that implement robot motion planning in real time. A summary of robot control architectures that describes the architectures used in mobile robotics are included to explain the different approaches to motion control and how this impacts motion planning. A brief summary of the literature describing the Pioneer robot and its operating system Saphira completes the related work overview.

1.2 Robot Motion Planning

The field of robot motion planning is a broad one, encompassing all robot manipulator applications in addition to the autonomous robot applications. The field can be divided by many characteristics, each sub-domain representing a conceptual method based on different assumptions and environmental conditions. The textbook of Latombe 1991 defines the separate classifications as cell decomposition, roadmap, and potential field methods. These broad definitions exclude the detailed implementations that accompany the specific methods and summarize from a conceptual level the nature of their operation. The ability to rapidly seek a path despite the holonomic and nonholonomic constraints is one advantage that this method of motion planning has over previous attempts like Find-Path by Brooks 1983 or Brooks and Lozano 1982.

These methods must employ search tree algorithms to explore the configuration space for viable paths that conform to the configuration space. As such the ability to search for an optimal path through R^3 configuration space must be measured by the computational efficiency despite the NP-complete problem. The gradient ascent method of the proposed non-learning ANN method has a computational size as large as the unique path from the initial configuration to the desired end configuration.

General Robot Motion Planning Methods

For the sake of conformity, the field is divided into techniques that conduct robot motion planning using either global or local methods. This refers to the amount of knowledge the robot has at the time of operation where global methods are assumed to have knowledge about the entire workspace and local methods do not. Generally, global methods involve some type of world modeling that stores information about the entire space. Local methods do not require complete knowledge of the workspace.

Global Motion Planning Methods

The fundamental assumption made by these techniques is that the robot has access to global knowledge of the environment. This key factor is rarely achieved in practice, but it is not an untenable assumption when considering that any method responds to changing environments. Therefore, if the global knowledge has uncertainty, these methods have the ability to correct the solution path under changing degrees of knowledge. Considering that most mobile robots with a sensor array can on focus on

the immediate surroundings of the robot, then the responsiveness of the robot to conditions will depend on the efficiency of the path planning method used.

Global methods assume an extensive and accurate knowledge of the robot's environment as a condition of path planning. This knowledge may include *a priori* model knowledge, such as a pre-existing map of the environment, or it may come in the form of total sensor coverage. Generally, global methods are employed where a high confidence exists in the availability and certainty of data from the environment exists. is an example of this method.

Sleuner and Tschihold-Gurman 1999, Van der Stappen, *et. al.* 1995 , Cherif *et. al.* 1994, Kortenkamp *et al.* 1994, Zimmer and von Puttkamer 1994, Quinlan and Khatib 1991, Latombe 1991, and Gouzene 1984 are examples of the cell decomposition method. In this method, the free space of the workspace in question is divided into rectangular regions that can be processed in a variety of manners. Exact cell decomposition methods decompose the free space into cells whose union is exactly free space. Approximate cell decomposition methods produce cells of a predefined shape whose union is contained within the free space. Adjacent cells share a common vertex or face within the space. Once the free space has been segmented into cells, a connectivity graph is developed for the search algorithm. These methods employ a search algorithm as the final approach to determining the path.

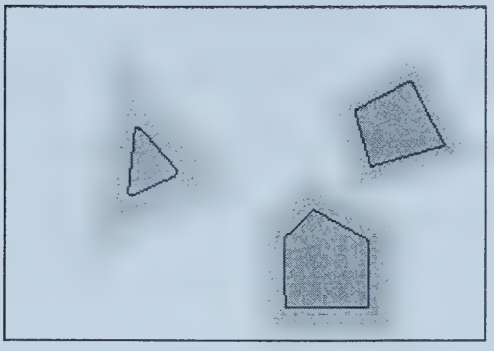


Figure 1.1: Potential field method generating field around obstacles

Krogh 1984 and Khatib 1985 are some of the first examples of the potential field method for robot motion planning. This method was also used by Hsu *et. al.* 1996, Bouilly et. al. 1995, Connolly *et. al.* 1994, Canny and Lin 1992, and Khatib 1986. This method assigns a value for the desired pose within the environment and then computes a vector field across the space towards the current pose of the robot. This method assigns a negative value to regions that contain an obstacle so that the resultant effect on the potential field in the area of an obstacle makes it less likely for a robot to attempt a path near the robot. The final path is determined as a gradient ascent search towards the desired position. One of the deficiencies of this method is the ability of the robot to become "stuck"- to oscillate between local maxima- in the region surrounding an obstacle in the direction of the desired final position, which is the global maxima. The proposed ANN method of Yang/Meng alleviates this problem by employing obstacles that have only local affect.

Roadmap methods are the third commonly developed methods in the field of robot path planning. There are five common versions of the roadmap method: Voronoi, silhouette, freeway, retraction, and visibility graph. The specific implementations employ various methods, but all these methods embody the connectivity of the free

space between obstacles by creating a series of corridors that decrease the computational overhead over many iterations by storing the roadmap as a generic starting point for path planning. The final plan consists of searching the connectivity of the roadmaps, and then plotting the path as a series of roadmap lengths connected sequentially. Kim *et al.* 1998, and Canny and Lin 1992 are examples of roadmap methods. Laumond and Simeon 1998, and Fekete *et. al.* 1995 are examples of the visibility graph method. Chen et. al. 1995, Rajasekaran and Ramaswami 1992, and Goodrich 1989 are examples of global Voronoi diagram approaches. Kim *et al.* 1998 uses a roadmap method to construct multiple paths in the workspace that are chosen as the final path due to input from the sensors and adjustments in the localization errors. Lambert *et al.* 2000, Laumond and Simeon 1998, Guibas *et al.* 1997, Kavraki and Latombe 1997, and Laumond 1994 are examples of a newer form of the roadmap method, the probabilistic roadmap (PRM). Under the PRM, a Voronoi diagram is constructed and then the translation and rotation requirements are constructed from a uniformly distributed random set of translations and rotations. Each of the proposed paths is then checked, step-by-step, against the obstacles in the space for collision. If a proposed path contains no collisions then the intermediate steps are computed to form the trajectory. For nonholonomic robots or robots with geometrical complexity, the roadmap method by itself is not suitable where the roadmap does not contain enough connectivity to offer a full range of configurations. This is the case in environments with many obstacles or those with disjoint areas of free space. Laumond *et al.* 1994 describes the use of visibility graphs in a probabilistic roadmap method. One attempt to solve this was developed by Foskey *et al.* 1998 in which a hybrid system of a

generalized Voronoi diagram (GVD) and a PRM planner was used together for complete motion planning. The PRM, which is computationally expensive, is used only for regions where the GVD has estimated a collision and must be resolved by another method. This produces an increased speed path planner in comparison to a PRM alone. Papadimitriou et *al.* 1994 investigated a visibility graph method that completes an PSPACE-hard problem graph search of a global environment with obstacles at the vertices. This PSPACE-hard search is similar to a tree search problem. Lambert *et al.* 2000 addresses path planning using a roadmap model and Djikstra's uninformed search algorithm for car-like robots with sensor and localization uncertainty. This method uses landmark features within the roadmap as via points for the planning between initial and final configurations. This plan compensates for positional error that develops during travel. Features within the roadmap are examined with an uncertainty field to measure how well a feature is perceived at every instant of motion.

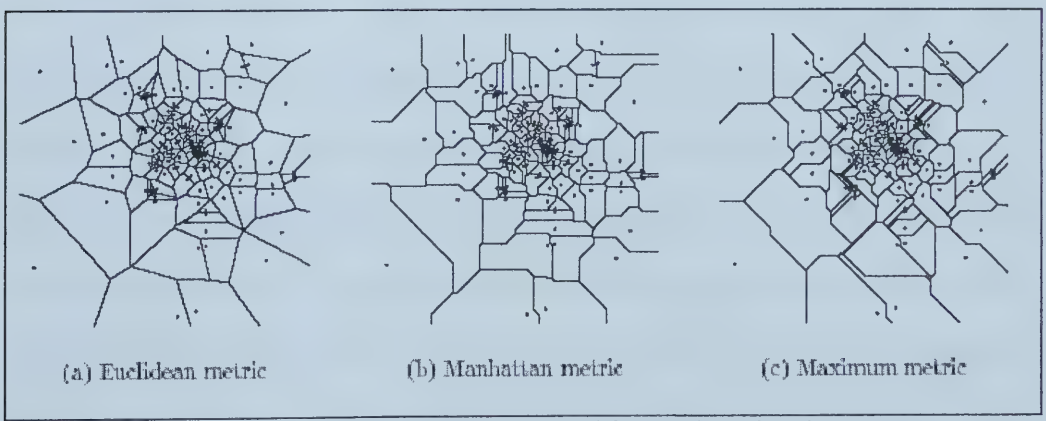


Figure 1.2: Voronoi map method for motion planning

Panel (a) using Euclidean distance metric, Panel (b) using Manhattan distance metric and Panel (c) using Maximum metric

Other methods for path planning have been used. Genetic Algorithms were used by Ahuactzin et *al.* 1999. These genetic algorithms were only able to achieve results with massively parallel machines. Qin and Henrik 1996 demonstrated a novel parallel motion planning algorithm for robot manipulators that used a parallel virtual machine to search randomized multiple paths under a Find Path algorithm concurrently and was suitable for individual robot manipulators in dynamic environments or multi-robot environments. It was shown to work for a 6-DOF (degree of freedom) manipulator modeling a PUMA 200 in several seconds. Cook 1995 demonstrates a parallel search algorithm based on heuristics for searching configuration space and guarantees an optimal solution of the path.

Local Motion Planning Methods

Local planning methods assume that the only knowledge of the environment that a robot can use with certainty is the information provided from sensor in a limited envelope around the robot. Latombe defines potential field as a local method and the cell decomposition and roadmap methods as global. If the amount and scope of information available is the issue, then one could utilize any of the above methods as local or global. Potential methods must use the global location of the desired final configuration if they are to escape local minima. Cell decomposition methods could be limited in free space segmentation if the scope of the cells is limited to a local region of the free space and the composite path plan was divided into smaller goals within defined space. In the field of local path planners, there are several methods of trajectory generation that employ a reactive path planner as opposed to the more

traditional deliberative methods. The global methods are all deliberative in that they plan the entire motion from initial to final configuration. Reactive methods, primarily the sensor-based methods, alter the path with intermediate segments en route so that the recent information from localization and sensor fusion can be incorporated in a real-time manner to increase the likelihood of success. Generally, local sensor based methods require no internal world model to complete robot motion, as the direct influence of the sensors results in an arbitrary change in path trajectory.

There are many heuristic-based motion planners as well. Smirnov *et al.* 1996 employed a heuristic combination of exploration and exploitation in their VECA Variable Edge Cost Algorithm to search for the goal state in a state space that is not completely known. This method makes the search for a goal state is more likely than what has been achieved with either method individually. Scheuer and Fraichard 1995 and Scheuer and Laugier describe planning continuous curvature paths using clothoid arcs and other primitive line segments.

Krogh 1984 used the potential field method in a local path planner. Other examples of potential field planners are Brock and Khatib 1998, Fox *et. al.* 1998, Bouilly *et al.* 1995, Borenstein 1989, Khatib 1987, and Khatib 1986. Bouilly *et al.* 1995 employed a potential field in a sensor-based motion planner that dealt not only with localization errors in the estimation of the current position, but also the use of motion primitives (pre-computed trajectory elements) that were used to patch areas of uncertainty. This method employed the potential field to construct an initial path assuming no uncertainty and then used the primitives to compensate for regions of uncertainty. This approach was able to achieve motion plans for a circular mobile

robot simulation in the range of a few seconds to tens of seconds. Borenstein 1989 describes a modified potential field method, called the virtual force field, which combines occupancy grids with a potential field method to make a more reliable navigation system

Stentz 1994 was an implementation of a visibility graph algorithm for path planning with local information and uncertainty. Mandelbaum *et al.* 1996 employed a hybrid deliberative/reactive path planner. It employed a feature grid to chart the location and extent of all obstacles detected by the sensors. This forms the foundation of motion around the obstacles but compensates with navigation adjustments to plot out the uncertainty in the environment.

Konolige *et al.* 1996, Ruspini *et al.* 1995, and Saffiotti *et al.* 1993 describe a navigation system based on behaviours using fuzzy rule sets that alters course in an iterative fashion as it moves towards the goal position. This architecture combines the goal-directedness of higher-level behaviours with lower-level reaction to the environment. This architecture is described alter on in the section devoted to Saphira. This method does not rely on complete or certain information, and does not rely on the data explicitly as it is presented from the onboard sensors. The overall architecture employs an augmented finite state machine that conducts map modeling (Local Perception Space), path planning in the form of course correction, and then trajectory execution.

Artificial Neural Network Robot Motion Planning Methods

Krose and van Dam 1996 describe applications of ANNs for both local and global planning methods. Thrun 1998 demonstrates the use of ANN techniques to construct Voronoi maps as well as other roadmaps to conduct path planning. Araujo *et. al.* 2000 demonstrates a two-layer recurrent Hopfield network for shortest path computation. Lagoudakis and Maida 1998, and Lagoudakis 1996 demonstrate a similar method to Yang and Meng 1999 using a Hopfield network that also can find a path in configuration space by a gradient ascent search. Knobbe *et. al.* 1996, Glasius *et. al.* 1995, Overmars *et. al.* 1995, Zimmer and von Puttkamer 1994, Sehad and Touzet 1994, and Vleugels *et. al.* 1993 are examples of a roadmap based technique using neural networks in a Kohonen Self-Organizing Maps.

This thesis presents the implementation of the Yang/Meng non-learning ANN algorithm for robot motion planning for the advanced motion problem. This algorithm is described in Yang 1999, and Meng and Yang 1999. This algorithm was based on the work of Hodgkin and Huxley (1952) in their research of the membrane model for a biological neural system and Grossberg's (1973) shunting model. Yang/Meng's ANN approach was presented in this application first by Yang (1999) in his work in robot motion planning. Yang's algorithm is an extrapolation of an exact cell decomposition technique that Latombe 1991 theorized for configuration space that uses a search algorithm to find the optimal/possible robot path. By Yang's method, the space search problem becomes a computational optimization problem and the final path is simply a gradient ascent/descent search through the network from initial to final pose. Yang's

work showed the theoretical foundation of this new approach and the possible applications in a variety of robot motion planning scenarios. The global motion planner presented here must be viewed as a sub-component of a larger system. This system separates the motion problem into path plan and trajectory generation problems. The path is a rough set of intermediate poses that align the centre of neurons representing a grid square. The trajectory generation is handled by another component that uses the path plan as a guide.

1.3 Robot Control Architectures

All robot control architectures are based on variations of the Sense-Plan-Act paradigm as described in Arkin 1998. This paradigm describes the linkage between the robot and its environment from the interpretation of sensed information to response. Figure 1.3 illustrates the paradigm used by Konolige 1997 in the Saphira operating system.

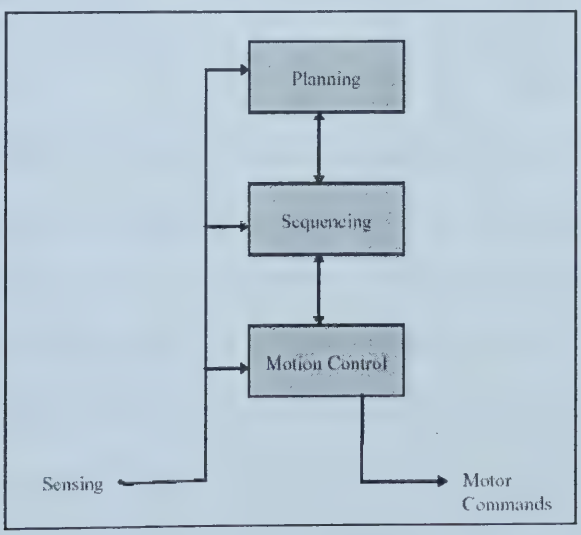


Figure 1.3: Sense-Plan-Act paradigm of Saphira

(permission for reproduction granted by Prof. Konolige and SRI International Ltd.)

From this paradigm, three different control architectures can be established based on reactive control, deliberate (or hierarchical) control or a hybrid of deliberative and reactive control. Figure 1.4 below describes the spectrum of deliberative to reactive control architectures as described by Arkin 1998. Konolige *et. al.* 1996 describes the Saphira operating system as a reactive control architecture.

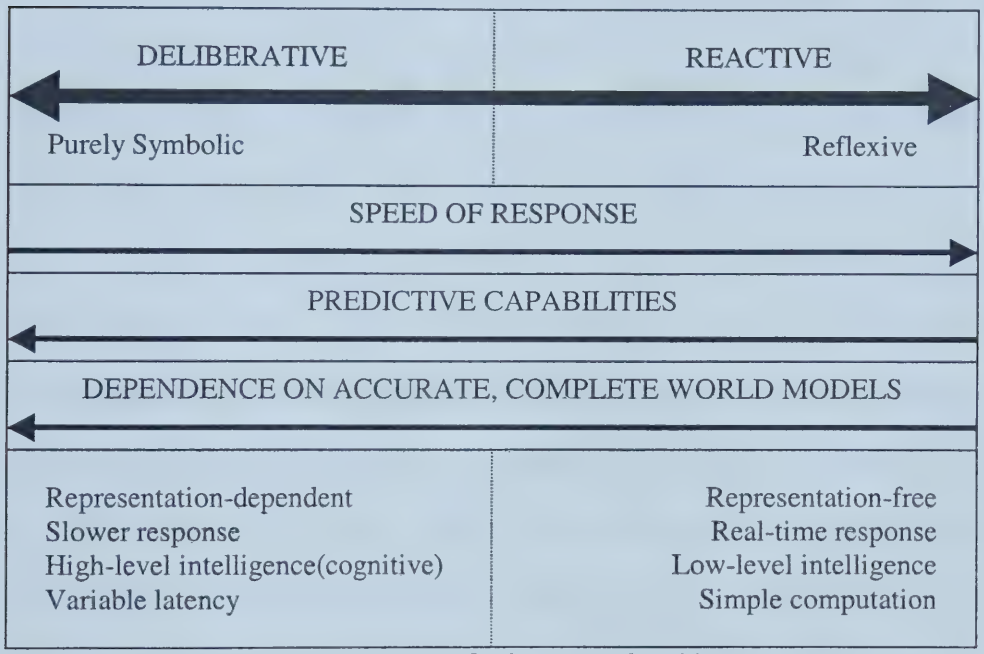


Figure 1.4: Spectrum of robot control architectures

(permission granted for reproduction from Prof. Arkin)

Behaviour-based control architectures were developed by Brooks 1985. Examples of a behaviour-based control architecture may be found at Kortenkamp *et. al.* 1998, Rosenblatt 1997, Kortenkamp *et. al.* 1994, and Langer *et. al.* 1993. This thesis proposes a hybrid control architecture by employing the ANN as a motion planner superimposed on the reactive behaviours in the Saphira system. This hybrid system is

similar to the Saphira operating system using a goal-directed logical language PRS-Lite on top of the behaviour-based robot control architecture to allow the incorporation of deliberative processes with the reactive behaviours.

Pioneer Robot and the Saphira Operating System

This thesis implements the ANN motion planning method on the Pioneer 2 DX. The Pioneer is described in the Operations Manual. The Saphira operating system is described in the Saphira Manual, Konolige, *et al.* 1996, Konolige and Myers 1996, Konolige 1996, Ruspini, *et al.* 1995. Saffiotti *et al.* 1999 and Saffiotti *et al.* 1993 describes the core fuzzy behaviour-based reactive robotic control architecture which proposed combining purely reactive behaviours combined with less imminent goal-directed behaviours to achieve more strategic goals. Saffiotti *et al.* 1993 illustrates the local minima problem with potential field methods like the fuzzy desirability function used in the reactive behaviours. Konolige 1996 explains the implementation of the Saphira operating system within the Open Agent Architecture and how this won the 1996 AAAI Robotics Contest.

1.4 Scope of Thesis

This thesis implemented the ANN motion planning method in MATLAB simulations as well as C++ within the Saphira operating system to examine its characteristics and performance in robot motion planning. The optimization of the dynamic neuronal activation model required research into finite difference approximation methods.

These methods were needed to effectively simulate the 1st order ordinary differential equations in order to achieve fast solutions when incorporated into Saphira. In addition to the ANN motion planner, the Saphira behaviour-based control architecture was modified to interface to the ANN motion planner. The complete system was then tested in experiments with the Pioneer simulator to evaluate the performance of the ANN motion planning method.

1.5 Thesis Objectives

The objectives of this thesis are as follows:

1. The primary objective is to explore the theoretical aspects and prove through implementation and results that this method conforms to the real-time performance requirements and stability issues for the Pioneer robot under the Saphira operating system;

2. Secondly, the implementation must judge the relative performance of this method to achieve a successful motion path under the various extensions of the advanced motion problem (as defined by Latombe 1991); and

3. Thirdly, this thesis explores the viability of this method adapted to other extensions of the advanced motion problem.

1.6 Organization of Thesis

Chapter 1 presented the related work and formal outline of this work. Chapter 2 lays out the problem definition and presents the underlying theory in motion planning. Chapter 2 also addresses the computation complexity of search algorithm such as the proposed ANN motion planning method. Chapter 3 presents the mathematical foundation associated with the non-learning ANN techniques and finite difference equations (FDE) issues surrounding the implementation of the software. Chapter 4 describes the Pioneer robot and the Saphira operating system. Chapter 5 outlines the implementation of the ANN motion planning method and the proposed hybrid control architecture for operation of the Pioneer robot. Chapter 6 presents the experimental results of this proposed method. Finally, Chapter 7 summarizes the thesis and discusses the objectives.

Chapter 2

Problem Definition

The definitions for the basic and advanced motion planning problem are presented here. The related definitions of workspace, obstacles, configuration space, and kinematic constraints are also dealt with in this chapter. Latombe 1991 provides a *de facto* standard for definitions in the field of robot motion planning presented in this chapter. Computational complexity for motion planning problems is discussed at end of this chapter.

2.1 Definitions

Foremost in this thesis must be the definition of workspace. Let W be a Euclidean space represented as R^n where n is either 2 in the case of a 2D workspace or 3 in the case of a 3D workspace where the robot motion must be planned. W is the universe of discourse for the path-planning problem and contains all objects/obstacles and the robot itself within it. W is the workspace and $\{0\}$ is the global reference frame of the workspace.

Let $\Im$ represent the subset of W that contains all the space not currently occupied by objects. Equation (2.1) describes $\Im$ as contained in W. $\Im$ is the free space of W.

$$\Im \subset W \tag{2.1}$$

An obstacle O is an element of the space W that exists in the space. Space occupied by an obstacle O cannot contain other objects within it. The subset of W that is the union of all obstacles within W is the space that a robot cannot occupy in while in transition over a path. Let O_i represent an obstacle as in equation (2.2).

$$O_i \in W, \text{ where } i = 1,\ldots,M \tag{2.2}$$

M is the total number of obstacles in the workspace W. Obstacles are contained in W.

$$\{O_{1,\ldots,}O_M\} \subset W \tag{2.3}$$

Let the robot in question be represented by R, where $R \in W$. The robot R is said to be a free-flying object for the basic motion problem, which means that it has no constraints on motion from one point in W to another provided these points are within $\Im$. A free-flying robot, also known as a point robot, can achieve any arbitrary velocity along the x-, y- or z-axis (in the case of 3 dimensional workspaces) to accomplish any transition. The robot R is a rigid object and the geometry of each point within the object is fixed with respect to the robot coordinate frame {R}.

Given the above objects as the scope of the environment, the universe of discourse is the union of R, O_i, and $\Im$ which spans the workspace W. This union is

presented in equation (2.4).

$$(\Im \cup R \cup O_i) \subseteq W \tag{2.4}$$

It is useful to examine the basic motion problem in terms of the workspace and the transition from initial pose to the final pose. But another more valuable method was modified for robot planning by Lozano-Perez and it deals with the degrees of freedom of the workspace as a space within which the unique position and orientation of every point is defined. Let C be defined as the configuration space of W. The configuration space has a one-to-one correspondence to poses within the workspace. A configuration space C for a robot in the 2D workspace is represented by a set of 3-vectors consisting of x, y, and θ where θ is the orientation angle of the robot reference frame {R} with respect to the workspace reference frame {0}. Let this 3-vector be defined as the configuration q. Configuration q is defined in equation (2.5). Within this thesis, the terms pose and configuration will be used interchangeably.

$$q = (x, y, \theta)^T \tag{2.5}$$

Planning of motions within configuration space is valuable for advanced problems where the kinematic constraints of robots must be considered. Obstacles within the workspace W will occupy all configurations associated with their positions (along the θ axis) within the configuration space. This limits the size of free space for the motion of robot A near obstacle B_i as illustrated in Figure 2.1.

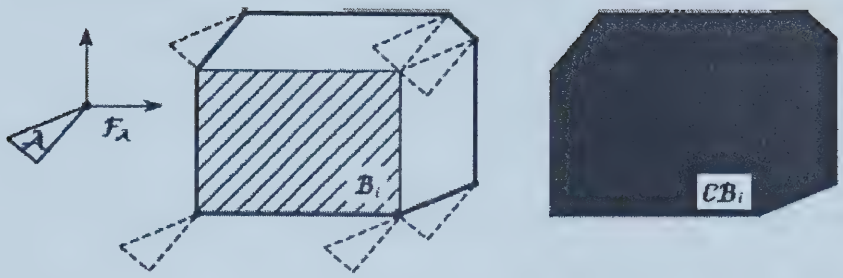


Figure 2.1: Obstacles in configuration space

(permission granted for reproduction by Prof. Latombe)

The path, as defined as by Latombe, is based on a topology in the configuration space using the distance metric that maps a change in the pose as a decrease in the distance between the current pose and the desired pose. All paths must, to be successful, decrease the distance over time in a continuous manner. The continuous requirement is a necessary condition of the path, where robots are not capable of executing discontinuous motions during transition. From the perspective of the physical implementations, real world systems will not be able to withstand adverse trajectories with discontinuities.

Let τ be the path of R, within the free space $\Im$. Let d be a distance metric that maps C onto R as in equation (2.6).

$$d : C \times C \rightarrow R \qquad (2.6)$$

The distance metric d is a continuous function of configuration, an n-vector in the configuration space C. Equation (2.7) defines the distance function as the Euclidean metric between any two points in the space. Let F define a region around configuration q within W. The distance d approaches zero as the current region of configuration q

approaches the region of the final configuration.

$$d(q_{current}, q_{final}) = \max_{\alpha \in F} \| a(q_{current}) - a(q_{final}) \| \tag{2.7}$$

A path of R from current to final configuration is therefore a continuous mapping of configurations within C. Let τ be defined as the path. The path τ is defined in equation (2.8).

$$\tau : [0,1] \rightarrow C \tag{2.8}$$

The configurations for current and final configurations as defined in equations (2.9) and (2.10) respectively.

$$\tau(0) = q_{current} \tag{2.9}$$

$$\tau(1) = q_{final} \tag{2.10}$$

2.2 Basic Motion Problem

Given the formal definitions above, the basic motion problem can be developed. In the basic motion problem, the knowledge of the positions and orientations and the precise geometries of all objects within the workspace are known. In the context of the previous chapter, this is referred to as global knowledge.

Given a robot R in a workspace W, where the precise initial position and orientation of R is known, and given a set of obstacles O_i within W, whose precise positions and orientations are also known. The obstacles are static within W. There exists a desired end state of R within the free space $\Im$ that is represented by an arbitrary

final position and orientation. The basic motion problem is to generate a path τ, if possible, from the initial pose of R to the final pose desired through $\Im$ within W. This path must be continuous and must only include intermediate poses that do not conflict with the obstacle set O_i within W. This path τ is a set of intermediate configurations within $\Im$ that must be followed in sequence in order for the robot R to achieve the final configuration. In the basic motion problem, the robot can achieve any final configuration from any initial configuration provided the free space between these points is continuous. Otherwise, there does not exist a path solution to this problem. An example of when a path does not exist would be if robot R, whose initial pose was on one side of an obstacle wall, were attempting to move to a configuration on the opposite side of the wall. This wall would have to be long enough to span the distance between the current and final configurations. Any path planning method must be equally capable of determining when a path is not possible as well as plan motions that can be achieved. The solution of the basic motion problem is not realistic, but it has been used to approximate the motions of some mobile robots. However, understanding the dynamics of this problem does lay the foundation for the advanced motion problem.

2.3 Advanced Motion Problem

In general, robot motion planning requires a less ideal set of assumptions to be applied to the basic motion problem. Any motion problem that assumes less rigorous

assumptions is an extension to the basic motion problem. These extended problems are referred to as the advanced motion problem. According to Latombe 1991, the following are elements that can be introduced to the basic motion problem to increase the complexity and realism of the problem:

Moving Obstacles

The introduction of multiple moving objects to the basic motion problem makes the solution of the path a time-varying function. Whereas the basic motion problem was time-invariant, the geometric path must now have a time component in the solution. Generally, the configuration space C can be extended to a time-varying version, CT, where the configuration space at each instant of time represents the current locations and orientations of all objects in the workspace. As the direction of time is invariant, the time axis must always increase during forward progress through the workspace. In practical terms, this translates to an iterative path planning problem where the plan at one point in the planning cycle may become impractical, or that new alternatives paths may appear in the CT space and the motion planner must be able to compensate for the changing conditions in real-time. The more realistic time-varying problem exists when the robot has kinematic constraints, where the motion of obstacles may impede the ability of the robot to compensate for the changing environment.

Multiple Robots

An extension of the moving obstacle problem includes multiple robots operating in the

same workspace and this presents a more complex situation for the individual robots. Obstacles are by definition not under the influence of the robot motion planner, and must be avoided at all costs. There arises a situation if multiple robots are present that the robot motion planner could coordinate the motions of several robots. This introduces the problems of scheduling and task prioritization in addition to obstacle avoidance. Another point to consider is that the interaction of multiple robots may negate viable generated motions when it comes to using the same free space (corridors) to traverse several independent and dispersed paths. Latombe points out two methods to deal with multiple robots in the workspace, a centralized motion planner that constructs a composite configuration space that amalgamates the configuration spaces of the individual robots and executes planning concurrently. The other method he developed was the decoupled motion planner, where each robot acts independently and plans are developed concurrently. When the robots come within range to affect each other, the motion planner then executes a combined motion planning method. This method speeds up the process of motion planning, but at the expense of completeness.

Uncertainty in the Environment

Another given assumption in the basic robot motion problem is that the workspace is known with certainty. The introduction of error and uncertainty decreases the effectiveness of arbitrary motion planning methods to complete otherwise possible configurations through a lack of knowledge about the workspace. Local motion planning methods are a subset of the environmental uncertainty set of problems, dealing with motion planning over small segments of the workspace without global

knowledge/scope. This is the real condition of most mobile robotic applications, where the range of sensors and the error introduced by sensor variance causes the information to be less than perfect. And in the case of environments with moving obstacles, the time-varying nature of the environment implies that areas not currently under sensor observation must remain suspect. This is important even when the mobile robot has just moved from an area and has stored artifacts about that region and incorporated them into the world model. Local knowledge or uncertainty affects the performance of motion planning algorithms.

Kinematic Constraints

Kinematic constraints, holonomic and nonholonomic, are manifestations of real world robots. A free-flying robot does not exist in reality. In general, through design or mechanical characteristics, all real robots suffer from constraints that limit the range of motions possible, or in the case of mobile robots limit the range of configurations that can be achieved from one pose to another desired pose. Holonomic equality constraints are defined as equality relations among the robot parameters in which the solution involves one of the parameters. With multiple independent holonomic constraints, the number of constraints reduces the dimension of the configuration space by an equal number. A holonomic constraint is an integrable relation among the parameters of C that can be solved for one parameter in terms of the others. In mathematical terms, a holonomic constraint "fixes" the other parameters by denying the full range of solutions as viable solutions of the equality/inequality. In practical

terms, a holonomic constraint limits the range of motions/orientations possible by the robot. One example is a robot in a 2D workspace that can translate freely about the workspace, but its orientation is fixed with respect to the workspace frame of reference. Latombe describes this as a particular case of the basic motion problem. An analogous example is the prismatic robot arm joint, where translation is possible along the axis, but no rotation is allowed. Nonholonomic inequality constraints are inequalities in the parametric solution involving the parameters and their derivatives. In the most widely known example, the car-like robot (whose dynamics are equivalent to a four-wheeled automobile also referred to as Ackerman steering) has its x and y velocities "locked together" within the workspace reference frame and dependent on the angle of the steering wheels. While a nonholonomic constraint does not decrease the dimension of the configuration space it restricts the dimension of possible differential motions from the current configuration.

2.4 Computational Complexity of Search Problem

The computational complexity of the basic motion planning method was determined by Reif 1979 that the lower bound for planning free motions in a configuration space of arbitrary dimension is PSPACE-hard. It was shown by Hopcraft, Joseph, and Whitesides 1984 that the lower bound for planning problems consisting of robot arms with revolute joints consisting of one-dimensional rigid planar links. Robotic systems with an arbitrary multiple rectangle shape were shown to be PSPACE-hard by Hopcraft *et al.* 1984 and PSPACE by Hopcraft and Wilfong, 1986. The planar arm with arbitrarily many links was demonstrated to be PSPACE-hard by Joseph and Plantiga,

1985.

For the cell decomposition method, which the Yang/Meng algorithm most accurately resembles, the upper bound on computational complexity was determined by Schwartz and Sharir as:

A free path in a configuration space of any fixed dimension m, when the free space is a set defined by n polynomial constraints of maximal degree d, can be computed by an algorithm whose time complexity is exponential in m and polynomial in both n ("geometrical complexity") and d ("algebraic complexity").

The work of Schwartz and Sharir implemented the Collins decomposition algorithm in their exact cell decomposition method. It was shown that this method has a complexity of twice the exponential of m. Canny 1998 produced a roadmap method that was singly exponential in m. The above research shows that there still does not exist a practicable solution with the previous methods for a real-time performance. Latombe 1991 predicts that, in general, the computational complexity of the robot motion planning problem increases exponentially in proportion to the dimension of the configuration space.

For the advanced problem with moving obstacles, Reif and Sharir 1985 determined the computational complexity as:

Planning the motion of a rigid object translating without rotation in three dimensions among arbitrarily many moving obstacles that may both translate and rotate is a PSPACE-hard problem if the velocity modulus of the object is bounded, and an NP-

hard problem otherwise.

Planning of a point robot in a plane, with bounded velocity modulus, among arbitrarily many convex polygonal obstacles moving at constant linear velocity without rotation has been shown to be NP-hard by Canny 1988. And Canny and Reif 1987 determined for motion planning with uncertainty that:

Planning compliant motions for a point in the presence of uncertainty, in a three-dimensional polyhedral configuration space with an arbitrarily large number of faces, is a non-deterministic exponential hard time problem.

In comparison to other methods of path planning, it is necessary to use the computation complexity of the search algorithms when compared to the non-learning ANN.

Chapter 3

Non-Learning ANN Approach to Robot Motion Planning

This chapter presents the mathematical foundation of the non-learning ANN method for motion planning on the Pioneer robot. This chapter outlines the relevant background of the non-learning ANN techniques for robot motion planning as described in Yang 1999. The end of this chapter describes the special numerical method considerations entailed in this application, and then finishes with the stability analysis of the non-learning ANN techniques.

3.1 Model of the Non-Learning ANN Neuron

Yang/Meng's method is an extension of the configuration space search as proposed by Latombe 1991. In this method, D denotes the entire 2D workspace of the robot. This planner decomposes the space D into a 2-dimensional array of small rectangular cells

of equal size. Each one of these rectangular cells is a node in the search graph G whose construction conforms to the nonholonomic constraints that are associated with the robot. Each of the neurons in Yang/Meng's method represents a central point in the square (or perhaps rectangular) region that is has arbitrary dimension in the x- and y- directions in relation to the robot. The singular difference is that the search graph is replaced by the gradient descent or gradient ascent search of the ANN neurons whose unique activation value for each node will describe a path, if possible, from initial to final configurations. In Latombe's method, a search is carried out that entails searching from every location along the path to other neurons that occupy free space and the search ends when a suitable path that ends at the desired final position is found. Latombe proposed using Djikstra's uninformed A* algorithm as can be found at Baase 1993. Yang/Meng's algorithm requires a single search that is computationally more efficient than Latombe's.

The robot motion planning method proposed is biologically inspired artificial neural network that can plan motions in a nonstationary environment. The model requires no learning or any other initialization other than connection of the neurons to one another according to the geometry of the workspace that it models. The constructed neural network represents the free space and obstacles within the workspace as neural activity. Target neurons attract globally while obstacle neurons repel locally. Neurons that represent occupied space differ from neurons representing free space and therefore a path through free space from initial pose to final pose can be determined.

Each neuron has an activation level that is derived from the voltage dynamics

of a neuron membrane as determined by the work of Hodgkin and Huxley 1952. Equation (3.1) describes the voltage dynamics across a neuron membrane.

$$C_m \frac{dV_m}{dt} = -(E_p + V_m)g_p + (E_{Na} - V_m)g_{Na} - (E_K + V_m)g_K \tag{3.1}$$

V_m is the voltage across the membrane. C_m is the membrane capacitance. E_K, E_{Na}, E_P are the Nemst potentials of Potassium, Sodium and the passive channels (leak current) respectively. The variables g_K, g_{Na}, g_P represent the conductances of Potassium, Sodium and the passive channels (leak current) respectively.

The shunting model of Grossberg 1973 is a simplification of the Hodgkin & Huxley model. This model is used as the basis for the non-learning neurons activations in this thesis. With the following substitutions, denoted in equations (3.2) through (3.8), equation (3.9) is reached.

$$C_m = 1 \tag{3.2}$$

$$\xi_i = (E_p + V_m) \tag{3.3}$$

$$A = g_p \tag{3.4}$$

$$B = (E_{Na} + E_p) \tag{3.5}$$

$$D = (E_K - E_p) \tag{3.6}$$

$$s_i^+ = g_{Na} \tag{3.7}$$

$$s_i^- = g_K \tag{3.8}$$

$$\frac{d\xi_i}{dt} = -A\xi_i + (B - \xi_i)s_i^+(t) - (D + \xi_i)s_i^-(t) \tag{3.9}$$

The variable ξ_i is the neural activation level of the i^{th} neuron. The designation i may represent any integer number. A, B, and D respectively represent nonnegative constants. A is the passive decay rate of the neuron's activation. B represents the upper bound of neural activity and D represents the lower bound of neural activity. The variables $s_i^+(t)$ and $s_i^-(t)$ represent the positive and negative inputs to the i^{th} neuron in the network.

From equation (3.9), $s_i^+(t)$ and $s_i^-(t)$ are expanded to include the representation of a target or obstacle for the i^{th} neuron as well as the input from adjoining neurons. Equation (3.10) is the neural activation equation used in the robot motion planning non-learning ANN presented by Yang 1999.

$$\frac{d\xi_i}{dt} = -A\xi_i + (B - \xi_i)([I_i]^+ + \sum_{j=1}^{N} \omega_{ij}[\xi_j]^+) - (D + \xi_i)[I_i]^- \tag{3.10}$$

$[I_i]^+$ represents a target at the i^{th} neuron in the network and $[I_i]^-$ represents the presence of an obstacle at the i^{th} neuron. The positive and negative inputs change over time and so the inputs can be more accurately represented as $[I_i(t)]^-$ and $[I_i(t)]^+$. These inputs are positive semi definite functions and are bounded over the interval [0,H), typically H>>A. $[\xi_j]^+$ represents the neural activation of adjoining neurons within the network. ω_{ij} is the connection weight between neurons ξ_j and ξ_i. ω_{ij} is a function of distance

as described in equation (3.11).

$$\omega_{ij} = f(d_{ij}) = \begin{cases} g & d_{ij} = 1 \\ 0 & otherwise \end{cases} \tag{3.11}$$

The gain factor g may be used to amplify the inputs to the current neuron. The distance d_{ij} is the Manhattan distance between neurons when arranged as a Cartesian grid on the x-y plane. Adjoining neurons are the only neurons that affect the activation of a given neuron. $[\xi_j]^+$ indicates the summation of activations from adjoining neurons only when those values are positive. When an adjoining neuron has a negative value, it is not added to the positive input. When a neuron represents an obstacle the neural activation over time will be a negative value bounded by -D. When the neuron represents the target location, then the activation value will be positive and approach B over time. All other neurons in free space will have a positive semi definite value reflecting their distance from the target neuron. Equation (3.10) was shown to be stable in the sense of Lyapunov by Yang 1999, and this will be presented later on in the chapter.

For this implementation, the proposed workspace map created by these neurons is a static geometric representation of the workspace. In other words, these neurons represent a fixed position on the plane of the workspace plane (assumed 2D). This is identical to the robot motion planning methods using occupancy grids. Each neuron, or grid square, represents a 2D area on the workspace plane and therefore has at most 8 adjoining neurons located around it as designated in Figure 3.1. Figure 3.1 describes the network topology used in this approach. The dimension N represents the length

and width of the grid area. Neurons at the edge of the workspace area have missing connections indicating the absence of adjoining neurons areas where robot motion is not possible.

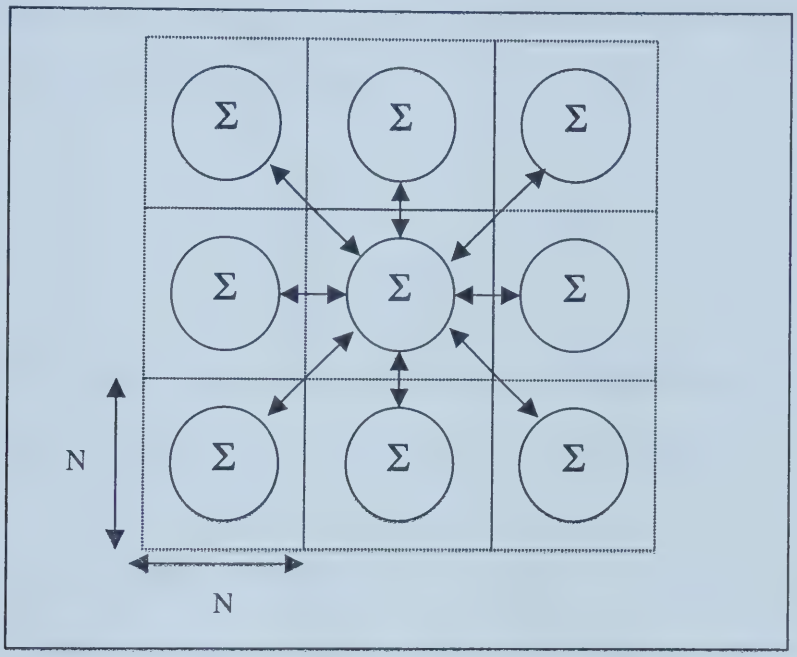


Figure 3.1: Single neuron linked to eight neighbouring neurons representing the occupancy within an area

The activation of each neuron in the network varies over time in accordance with equation (3.10). The presence of obstacles and a single target location alter the activation of the single neurons where they exist and these activations propagate to all other neurons in the network. The insertion and removal of obstacles and target locations over time allows this network to dynamically model the path from any neuron in the network to the target location while avoiding obstacles.

Searching the network using the gradient ascent rule will locate the target neuron from any free space neuron. The search will result in a single path if one exists.

The search rule requires the starting position and the activation of that neuron. Let p_c represent the current position, p_n be the next position in the network and p_t represent the target neuron in the network. The next position is found by comparing the activations in the adjoining neurons as per equation (3.12) where k is the number of neurons connected to the current neuron.

$$\begin{aligned} & p_n = \max \xi_j, \qquad j = 1,...,k \\ & if \quad p_n > p_c \, then \quad p_c = p_n \\ & if \quad p_c = B \, then \quad p_c = p_t \end{aligned} \tag{3.12}$$

This gradient ascent search is iterated until the target location is reached. The final motion involves moving through the neuron list from the current neuron in the plane to reach this target neuron. If a path does not exist then the activation of the adjoining neurons will not be greater than the current activation and the path will not reach a neuron with an activation value approaching the upper bound B. When the target location is blocked from the current position of the robot then the activations of all neurons near the current position will be zero. This characteristic of the non-learning ANN method insures that if a path exists, it is found by the gradient ascent search. This non-learning ANN method to robot motion planning reduces the computational complexity of the motion search to a linear value proportional to the number of neurons in the final path. However, in order to find the target neuron, the activation must be propagated throughout the network to the arbitrary current position of the robot without any *a priori* knowledge of obstacle locations. This propagation remains at a computational complexity of PSPACE, or proportional to a polynomial of

the number of neurons in the network.

The presence of an obstacle or a target impacts the final activation level of the i^{th} neuron significantly. In order to describe the dynamics of a target or obstacle at a given neuron, a simple simulation of a single neuron was conducted. Figure 3.2 depicts the time-varying activation level of a single neuron using Euler's approximation method with a value of $\Delta t = 0.00275$. The constant coefficients for this simulation were set at an arbitrary A = 10, B= 1, and D= 1 for this simulation. Initially, the activation is set to an arbitrary 0.15 but moves rapidly to the lower bound as an obstacle is present at the neuron at t = 0.00s. Before time approaches t = 0.01s, a target is inserted into the neuron and the resultant presence of a target negates the obstacle and the resultant activation approaches zero. After t = 0.15s, the obstacle is removed and the activation level approaches a positive upper bound. As can be seen in Figure 3.2, the activation level of the neuron over time moves towards the lower bound D when the obstacle is present and towards the upper bound B when a target is present. When they are both present, the activation approaches a steady state value of zero. The implication of a steady state of zero at a target neuron when an obstacle is also present is that no path will be found to it from any other neuron in the network.

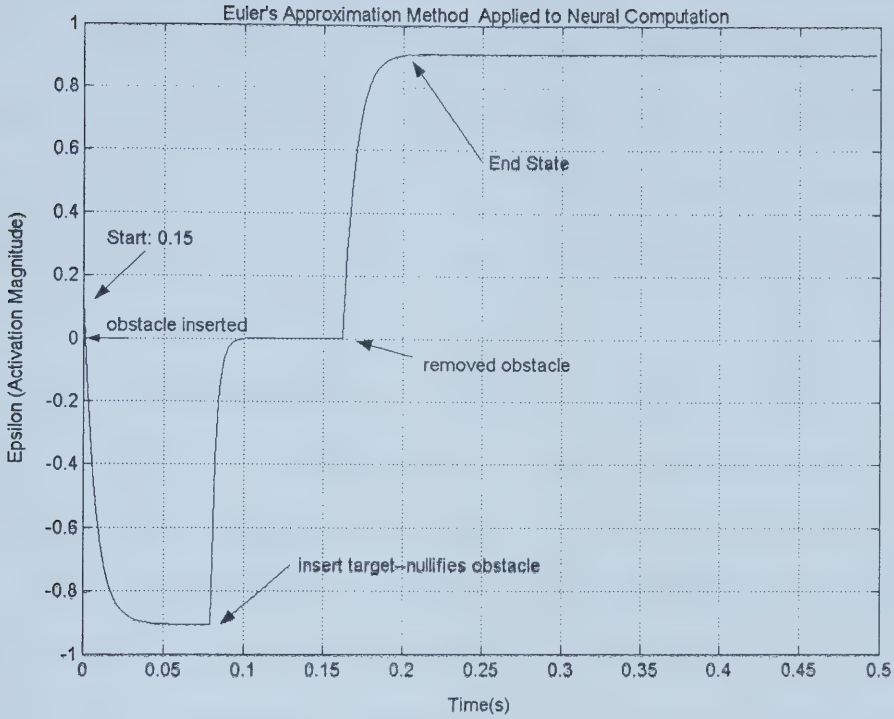


Figure 3.2: Activation level of a single neuron

3.2 Numerical Methods for Neuronal Modeling

As stated in the previous section, the propagation of the neural activations throughout the network remains a PSPACE problem. In order for the proposed method to be valuable in a real-time situation the network must approach a steady state quickly so that a new path, if it exists, may be determined and relayed to the robot for execution. The modeling of neural networks presents challenges to deriving a fast yet accurate solution. As we require the steady state activations for each neuron the problem is a one-dimensional initial value problem and so we have only one initial condition to assist approximation at each time step for each neuron.

The individual neuron models depend on the inputs from adjoining neurons and these inputs are linked to every other neuron in the network. As a result, the introduction of error into the approximation process at a single neuron affects the entire network. In addition, the finite difference approximation methods developed for time-marching or space-marching initial value problems exhibit the following errors according to Hoffman 1992:

1. Truncation errors;
2. Round-off errors; and
3. Inherited errors.

Errors in the initial data and algebraic errors are assumed to be non-existent. Yet the errors introduced for each neuron at the beginning of every computation include the error in the initial activation of the adjoining neurons. Therefore, errors in the initial data are present in the neuronal model. These sources of errors work against the process of finding fast and accurate network activation topography.

The location of obstacles and the target location are not known *a priori* and therefore the entire neural network must be updated at every iteration. The rapidly changing locations of obstacles and the current robot position require that the finite numerical method adjust quickly to the situation. The solution will take a considerable number of iterations to propagate from the target neuron and this is further increased by the number of obstacle neurons in the network. In order to determine if a path exists conclusively, the finite difference method must be carried out until a steady state solution is reached. This solution will vary in time taken with the complexity of the environment, where simple workspaces with limited obstacles may be determined

quickly while workspaces with more obstacles will require more iterations.

In order to implement and improve the modeling of the neural network for this thesis, several different finite difference approximation methods were applied and tested to determine the most effective for reaching a solution in all situations. Results of this examination are presented in Chapter 6. Implicit, explicit, and predictor-corrector single point methods were examined to determine the best process that balances the needs for speed against accuracy. Let $f(t_n,\xi_n)$ represent the first order ODE in equation (3.10). The time step for each method is represented by Δt. The single point methods chosen for this were as follows:

1. 1st order Explicit Euler Method;

$$\xi_{n+1} = \xi_n + \Delta t f(t_n, \xi_n) \tag{3.13}$$

2. 1st order Implicit Euler Method;

$$\xi_{n+1} = \xi_n + \Delta t f(t_{n+1}, \xi_{n+1}) \tag{3.14}$$

3. 2nd order Modified Heun's Method;

$$\xi_{n+1} = \xi_n + \frac{\Delta t}{2}(f(t_n, \xi_n) + f(t_n + \Delta t, \xi_n + (f(\xi_n + \Delta t)))) \tag{3.15}$$

4. 2nd order Modified Euler Predictor-Corrector Method;

$$\begin{aligned} \xi_{n+1}^P &= \xi_n + \Delta t(f(t_n, \xi_n)) \\ \xi_{n+1}^C &= \xi_n + \frac{\Delta t}{2}(f(t_n, \xi_n) + f(t_{n+1}, \xi_{n+1}^P)) \end{aligned} \tag{3.16}$$

5. 2^{nd} order Runge-Kutta Method (midpoint predictor-corrector);

$$\begin{aligned}
&\xi_{n+1} = \xi_n + C_1\Delta\xi_{n1} + C_2\Delta\xi_{n2} \\
&\Delta\xi_{n1} = \Delta t f(t_n, \xi_n) \\
&\Delta\xi_{n2} = \Delta t f(t_n + \alpha\Delta t, \xi_n + \beta\Delta t) \\
&C_1 = 0 \\
&C_2 = 1 \\
&\alpha = \beta = \frac{1}{2}
\end{aligned} \tag{3.17}$$

6. 3^{rd} order Bogacki –Shampine Method (from Shampine 1994); and

$$\begin{aligned}
&F_1 = f(t_n, \xi_n) \\
&F_2 = f(t_n + \frac{\Delta t}{2}, \xi_n + \frac{\Delta t}{2}) \\
&F_3 = f(t_n + \frac{3\Delta t}{4}, \xi_n + \frac{3\Delta t}{4}) \\
&\xi_{n+1} = \xi_n + \Delta t(\frac{2}{9}F_1 + \frac{1}{3}F_2 + \frac{4}{9}F_3) \\
&F_4 = f(t_{n+1}, \xi_{n+1}) \\
&error = \Delta t(\frac{5}{72}F_1 - \frac{1}{12}F_2 - \frac{1}{9}F_3 + \frac{1}{8}F_4)
\end{aligned} \tag{3.18}$$

7. 4^{th} order Runge-Kutta Method.

$$\begin{aligned}
&K_1 = \Delta t f(t_n, \xi_n) \\
&K_2 = \Delta t f(t_n + \frac{\Delta t}{2}, \xi_n + \frac{K_1}{2}) \\
&K_3 = \Delta t f(t_n + \frac{\Delta t}{2}, \xi_n + \frac{K_2}{2}) \\
&K_4 = \Delta t f(t_n + \frac{\Delta t}{2}, \xi_n + K_3) \\
&\xi_{n+1} = \xi_n + \frac{1}{6}(K_1 + 2K_2 + 2K_3 + K_4)
\end{aligned} \tag{3.19}$$

The local truncation error for the 1^{st} order methods is on the order of $O(\Delta t^2)$ and the

global error is on the order of $O(\Delta t)$. The local truncation error for the 2^{nd} order methods is on the order of $O(\Delta t^3)$ while the global error is on the order of $O(\Delta t^2)$. The local truncation error for the Bogacki-Shampine method is on the order of $O(\Delta t^4)$ and the global error is on the order of $O(\Delta t^3)$. The Runge-Kutta 4^{th} order method has a local error on the order of $O(\Delta t^5)$ and a global error is on the order of $O(\Delta t^4)$.

Lower order single point methods were selected to increase the total number of possible complete iterations through the entire network every second. Other methods of finite difference approximation using multi-point methods were not selected because the requirement to store previous values of the neurons activation would increase the storage requirements at a polynomial rate for larger neural networks. The overall accuracy was considered to be less important than the gradient between neurons so the gradient search method can find the path between neurons in the shortest time as opposed to waiting for a more precise solution to be computed by higher order methods.

3.3 Stability of ANN Method

It was shown in Yang 1999 that the non-learning ANN is stable in the sense of Lyapunov. For completeness, this property is presented here. From equation (3.10), the constants A, B, and D are positive. Typically, they assume values near 1. The inputs $[I_i]^+$ and $[I_i]^-$ are positive semi definite functions and are bounded to some region [0,H) where H is an arbitrary positive constant. The input of weighted activations from adjoining neurons is positive semi definite over the region bounded by [0,Bg) where g

is the gain factor. Based on the research of Yang 1999, the values for coefficients B and D are equal in order to bound the neural activities within a region of the origin. Consider the candidate Lyapunov function $V(\xi_i)$ which is positive definite in R.

$$V(\xi_i) = \frac{1}{2}\xi_i^2 \tag{3.20}$$

This Lyapunov candidate function is zero at $\xi_i = 0$ and positive definite within the domain of neural activations excluding the origin. The derivative of this candidate over the trajectories of equation (3.10) is given by (3.21).

$$\begin{aligned}\dot{V}(\xi_i) &= \frac{dV}{d\xi_i}\frac{d\xi_i}{dt} \\ &= \xi_i(-A\xi_i + (B-\xi_i)([I_i]^+ + \sum_{j=1}^{N}\omega_{ij}[\xi_j]^+) - (D+\xi_i)[I_i]^-) \\ &= -\xi_i^2(A_i + ([I_i]^+ + \sum_{j=1}^{N}\omega_{ij}[\xi_j]^+) + [I_i]^-) + B\xi_i([I_i]^+ + \sum_{j=1}^{N}\omega_{ij}[\xi_j]^+) - D\xi_i[I_i]^-\end{aligned} \tag{3.21}$$

It has been shown by Yang 1999 that $\dot{V}(\xi_i) < 0$ if the following relationship holds.

$$B \le \frac{\xi_i^2(A_i + ([I_i]^+ + \sum_{j=1}^{N}\omega_{ij}[\xi_j]^+) + [I_i]^-) + D\xi_i[I_i]^-}{+ B\xi_i([I_i]^+ + \sum_{j=1}^{N}\omega_{ij}[\xi_j]^+)} \tag{3.22}$$

Chapter 4

The Pioneer Robot and the Saphira Operating System

This chapter describes briefly the robot and its software used in this thesis to evaluate the ANN motion planning method. The Pioneer 2 DX robot's kinematic model, hardware, and specifications are presented in the following section. The software is presented at the end of the chapter to describe the existing robot control architecture.

4.1 Kinematic Model

The Pioneer 2 DX and the global and robot frames of reference are represented in Figure 4.1. The global frame of reference {0}, as shown in Figure 4.1, used with Pioneer and Saphira is rotated +90 degrees from the regular Cartesian coordinate system with the x-axis facing along the regular y-axis and the y-axis facing along the regular negative x-axis. The robot frame of reference {R} uses the x-axis as the along-

track axis and the y-axis as the across-track axis. The robot is depicted as it is represented in the Saphira GUI.

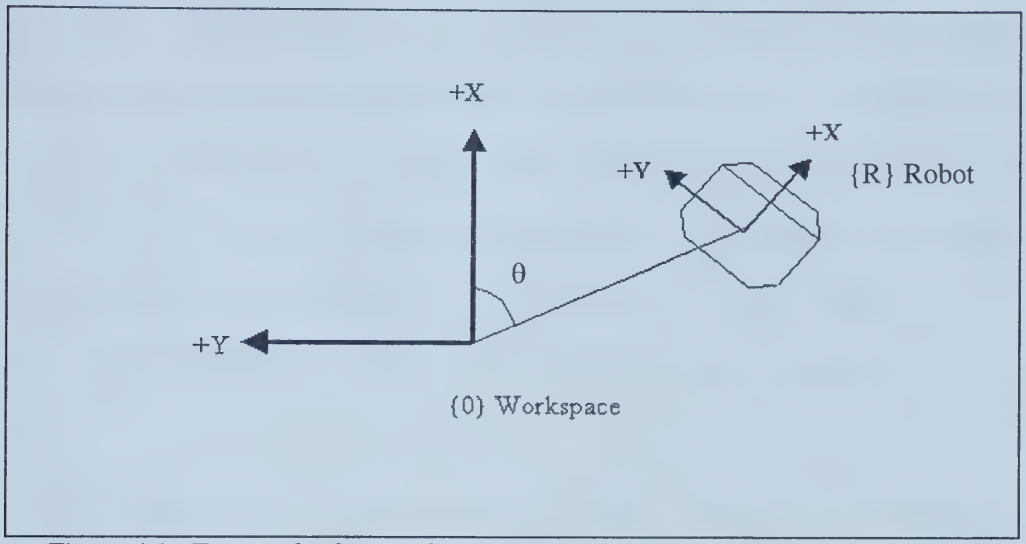


Figure 4.1: Frame of reference for workspace {0} and Pioneer 2 DX robot {R}

This robot has 3 degrees of freedom representing its Cartesian position (x, y) on the floor and orientation (θ) with respect to the global frame of reference {0}. Let q represent the 3-vector configuration of the Pioneer 2 DX robot as shown in Figure 4.1. The configuration q is referenced from the centre of the drive wheel axis. The derivative of q is useful for the state space model of the robot's motion. Equation (4.1) defines the derivative of q.

$$\dot{q} = (\dot{x} , \dot{y} , \dot{\theta})^{T} \tag{4.1}$$

The robot frame of reference {R} for R is taken from the global reference point q and the along track axis is defined as the x-axis and the across track axis is defined as the y-axis according to the right hand rule. Rotation of the robot around the reference point q is measured from the along track axis.

The Pioneer 2 DX has two independent drive motors attached to the front two

wheels set along the central axis of the chassis. There is a castor wheel located at the rear of the chassis for support, but this wheel is not driven and it is free to rotate in any direction. It is assumed that the drive wheels do not skid and that the velocities achieved are sufficiently small so that the robot does not slide in a lateral direction during forward/reverse motion. These assumptions simplify the kinematic model of the robot and are reasonable due to the friction coefficient of the rubber tires and the general condition of building floors where the Pioneer is designed to operate.

The kinematic equations for the Pioneer 2 DX are based on equation (4.2)

$$\dot{q} = K(q)v \tag{4.2}$$

where K(q) represents a state space model of the motion of the robot with respect to the control variables the drive wheel velocities. Let v represent the drive wheel velocity vector where $v \in R^2$. Therefore, v represents the left and right drive wheel velocities defined in equation (4.3).

$$v = (v_r, v_l)^T \tag{4.3}$$

Equation (4.2) can be rewritten as equation (4.4)

$$\dot{q} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{(C\theta)}{2} \\ \frac{(S\theta)}{2} \\ +\frac{1}{b} \end{bmatrix} v_r + \begin{bmatrix} \frac{(C\theta)}{2} \\ \frac{(S\theta)}{2} \\ -\frac{1}{b} \end{bmatrix} v_l \tag{4.4}$$

where b represents the distance parallel to the drive wheel axis from the centre of the left drive wheel to the centre of the right drive wheel. This mobile robot has 2 degrees of freedom (2DOF). This model was described in Zhao and BeMent 1992 as a two-

rear-drive-wheel vehicle. Zhao and BeMent 1992, showed that this model is weakly controllable or locally accessible.

The translational velocity in the along track direction of the robot is a function of the left and right velocities as in equation (4.5)

$$\bar{v} = \frac{(v_r + v_l)}{2} \tag{4.5}$$

The rotational velocity is a function of the left and right drive wheel velocities as well as the diameter b between the drive wheels shown in equation (4.6)

$$\dot{\theta} = \frac{(v_r - v_l)}{b} \tag{4.6}$$

The robot has a nonholonomic constraint, therefore the transition from an initial pose to any other configuration is constrained to a sub manifold of the configuration space. In other words, this constraint does not reduce the dimension of the configuration space but it does limit the possible differential motions within the 3 DOF configuration space. This non-integrable constraint can be derived from equation (4.4) and is shown in equation (4.7).

$$dyCos\ \theta = dxSin\ \theta \Rightarrow dyCos\ \theta - dxSin\ \theta = 0 \tag{4.7}$$

This nonholonomic constraint limits the translation in x and y directions to differential motions that satisfy this equation. As you will note the trivial solution of dy = dx = 0 does satisfy this constraint and yet does not restrict the differential rotation of the robot to achieve an arbitrary orientation. When the left and right velocities are the same magnitude while in opposite directions to one another then the translational velocity is zero. This is the basis for the turn radius of zero.

4.2 Hardware

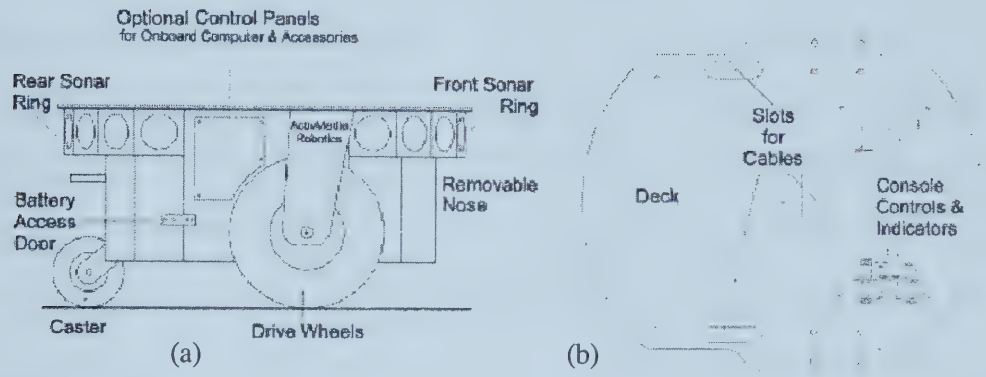


Figure 4.2: Pioneer 2 DX

Panel (a) Side View, Panel (b) Top View

(permission granted for reproduction by ActivMedia Robotics)

Figure 4.2 shows the Pioneer 2 DX robot. The Pioneer 2 DX robot chassis has the following specifications as laid out in Table 4.1.

Control of the motors is executed by the 20MHz Siemens 88C166 microcontroller. The microcontroller has 32KB of Flash ROM memory and 32K of dynamic RAM. It has 2 RS-232 interface ports and a 20-pin expansion port for general-purpose digital input/output, analog to digital input, and PWM ports. The microcontroller also has a watchdog timer that disengages the drive motors if the communications channel between host computer and robot server are severed for 2 seconds.

The Pioneer 2 DX has three sensor types working in concert to perceive information from the environment: bump rings, ultrasonic sensors, laser range finders,

and a visual camera. The first sensors are the bump rings of simple contact sensors that determine that the front or rear of the vehicle has collided with an object and the

Table 4.1: Pioneer 2 DX specifications

Length	44 cm
Width	33 cm
Height	22 cm
Mass	9 kg
Payload Mass	20 kg
Construction Material	CNC fabricated aluminium
Battery	3 x 12 V charge of 252 hours run time 8-10 hours
Drive Wheels	2 Solid Rubber wheels (16.5 cm dia. 3.7 cm width) with rear balancing caster
Position Encoders	9850 ticks per revolution at 76 ticks per mm
Turn Radius	0 cm
Swing Radius	min 26 cm
Translation Speed max.	1.6 m/sec
Rotation Speed max.	300 degrees/sec
Traversable Step max.	2 cm
Traversable Gap max.	8.9 cm
Traversable Slope max.	30% grade

approximate side of the vehicle that is involved. Bump rings on the front and rear have

5 independent sections. The microprocessor also monitors current draw to the two motors in order to infer situations where the motors are stalled as collisions.

The multiplexed ultrasonic sensor arrays of eight sensors are located on the front and back to detect objects up to 5 meters depending on the selected sensitivity. Ultrasonic sensors fire at 25 Hz. The current sensitivity of ultrasonic sensors is about 3m disregarding specular reflections. Sonar sensors fire in a default left to right pattern front and rear in order to lower probability of interference between individual sensors.

Figure 4.3 shows the configuration of the eight ultrasonic sensor arrays for the front and rear.

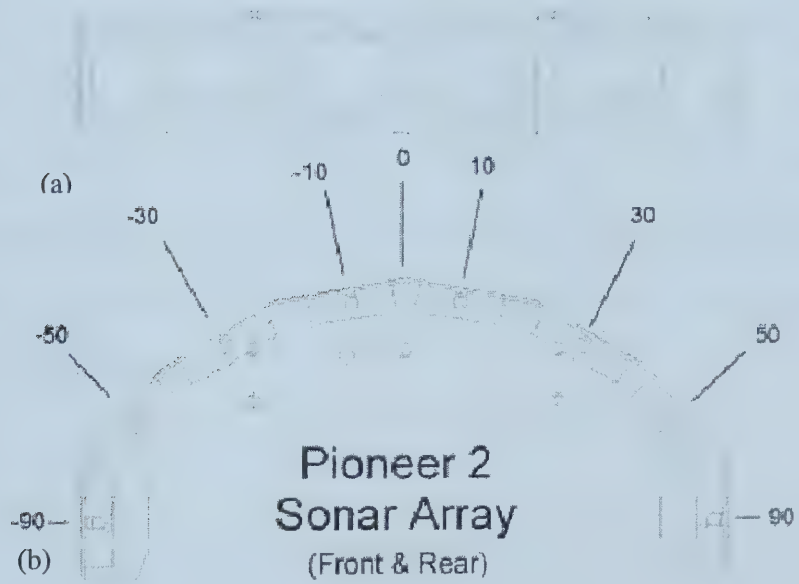


Figure 4.3: Sonar array layout

Panel (a) Front View, Panel (b) Top View

(permission granted for reproduction by ActivMedia Robotics)

Localization and navigation are assisted by the wheel encoders that count wheel revolutions. Given a constant wheel diameter and a non-noisy wheel encoder input (without encoder jitter), the current position can be derived as a function of the number of revolutions of the left and right drive wheels that have occurred since the previous localization. The wheel encoders have a resolution of 9850 ticks with 76 ticks per mm.

Communication with the Pioneer 2 DX can be carried out by RS-232 connection, onboard Ethernet modem, or attached laptop host computer. The Pioneer 2 DX is configured as a server to client applications on the host computer. The Pioneer 2 DX receives command packets from the host computer that contain instructions and sends server information packets (SIPs) that convey the current robot status, current pose configuration, left and right drive wheel velocities, battery power, bumper status, set point for angular velocity, pan tilt position, compass heading, sonar readings, and digital and analog inputs. Error checking involves a 2-byte checksum. SIPs are sent automatically and can operate at 10 or 20Hz. The communications protocol does not employ handshaking and so the packets may be missed if the communication channel drops out for any reason.

4.3 Saphira

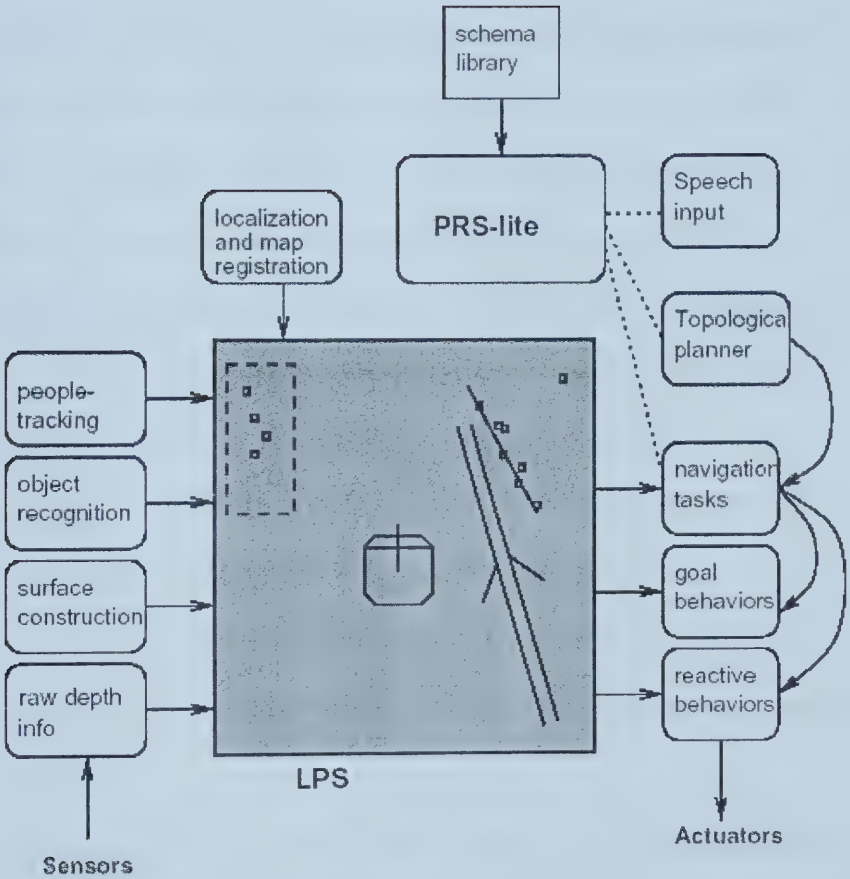


Figure 4.4: Saphira architecture

(permission granted for reproduction by Prof. Konolige and SRI International)

Saphira, according to Konolige and Myers 1996, is an integrated sensing and control system based on the local perceptual space (LPS) model, shown in Figure 4.4 above, that uses simple geometric representations of the space and obstacles within the space.

Saphira can be made to work with various modeling approaches (such as occupancy grids) depending on the application. The LPS ties together all sub functions and micro tasks that involve the many different concurrent processes that must cooperate to successfully control the robot. Saphira uses a behaviour-based architecture for control of the robot drive commands. The behaviour system uses a behaviour shell that executes at 10 Hz to update the current commands of the drive wheels. This shell executes all behaviours in ascending order of priority so that the most immediate responses to the environment (avoid collision) will always have precedence over the less immediate responses (navigate to next goal point). Behaviours are given a priority number and those of equal priority will be averaged using a fuzzy desirability function (see Saffiotti *et al.* 1993). When higher priority behaviours are activated by the environment, they will subsume or supersede the lower priority behaviours. This robot control architecture is reactive. Each behaviour has up to eight fuzzy rules that monitor the environment and infer the proper response for the robot motion. These decisions are defuzzified and given to the Pioneer robot as the translation and rotation velocity set points.

In addition to the behaviour-based control of the robot motion, several background function known as micro tasks are executed to monitor and interpret the LPS and update the communication between the robot server and the host computer. These functions use the sensor input to develop the LPS, localization and map registration for artifacts within the LPS.

Above the behaviours is PRS-Lite which manages behaviours in accordance with current tasks. PRS-Lite is an activity schema based on Procedural Reasoning

System task control. The procedural knowledge is represented as goal sets. These goal sets are a collection of goal statements that become satisfied as the ordered sequence of goals are satisfied. PRS-Lite was not used directly during the course of this thesis.

Saphira was incorporated into the Open Agent Architecture for use within multiple platforms. This Open Agent Architecture allows use in a distributed network including different agents with a simple "plug and play" protocol for introduction of additional resources into the system. Figure 4.5 demonstrates the graphical user interface (GUI) used for Saphira on the Windows32 systems.

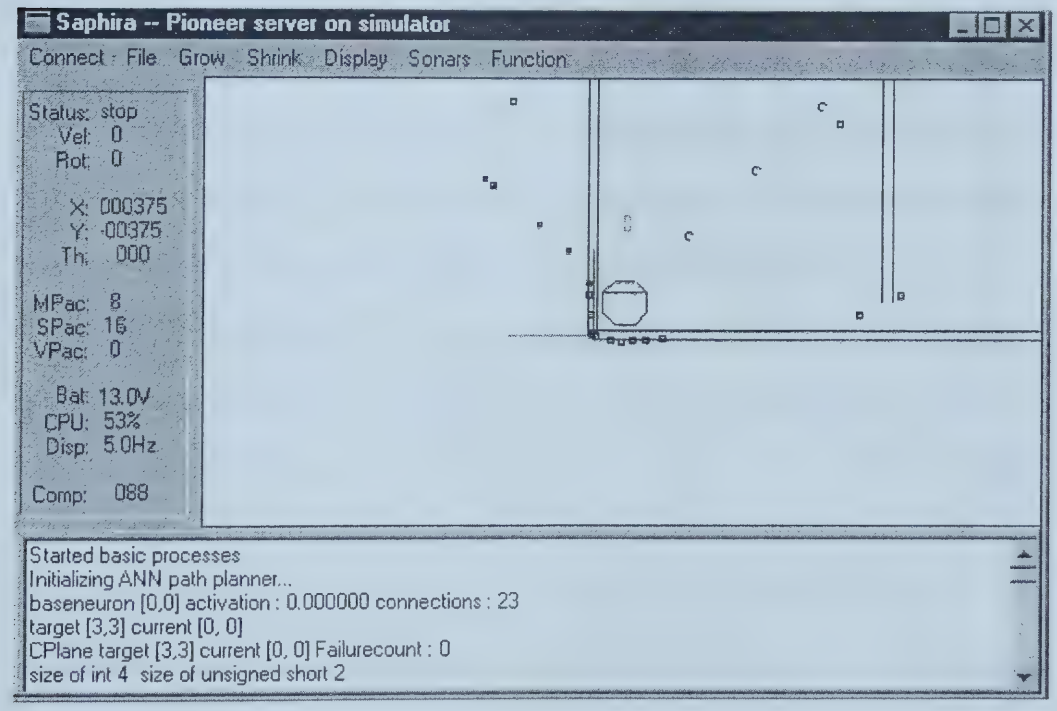


Figure 4.5: Saphira GUI interface

(permission granted for reproduction by Prof. Konolige and SRI International)

Chapter 5

Implementation

This chapter presents the implementation of the non-learning ANN for robot motion planning on the Pioneer robot. The first section describes the ANN implementation in C++ using the Seidewitz and Stark 1995 abstraction procedure. The second section describes the modified designed behaviours for the behaviour-based architecture of the Saphira operating system used to control the Pioneer robot. The chapter ends with a brief description of the software metrics involved in coding this thesis.

The implementation of this thesis was conducted in two parts. The first part of the implementation involved a study of finite difference approximation methods in order to determine the optimal neuron model that would provide a fast and reliable solution without *a priori* knowledge of the workspace. Several FDE methods were coded and tested against an evaluation workspace to determine this result. The procedure to process neuron models was examined and improved to increase the effect of the FDE's. In addition, this evaluation determined the optimal values for the constant coefficients A, B, D, and g.

The second part involved the design of the ANN Planner and hybrid robot control architecture and development of the object classes that would comprise the

final application. This application was used to test the Pioneer robot in two extensions of the advanced motion problem.

5.1 ANN Motion Planner Implementation

Figure 5.1 depicts a level 0 abstraction of the complete system with respect to the non-learning ANN motion planner. As an isolated function, the ANN motion planner must receive information from the Saphira operating system and relay the path back to

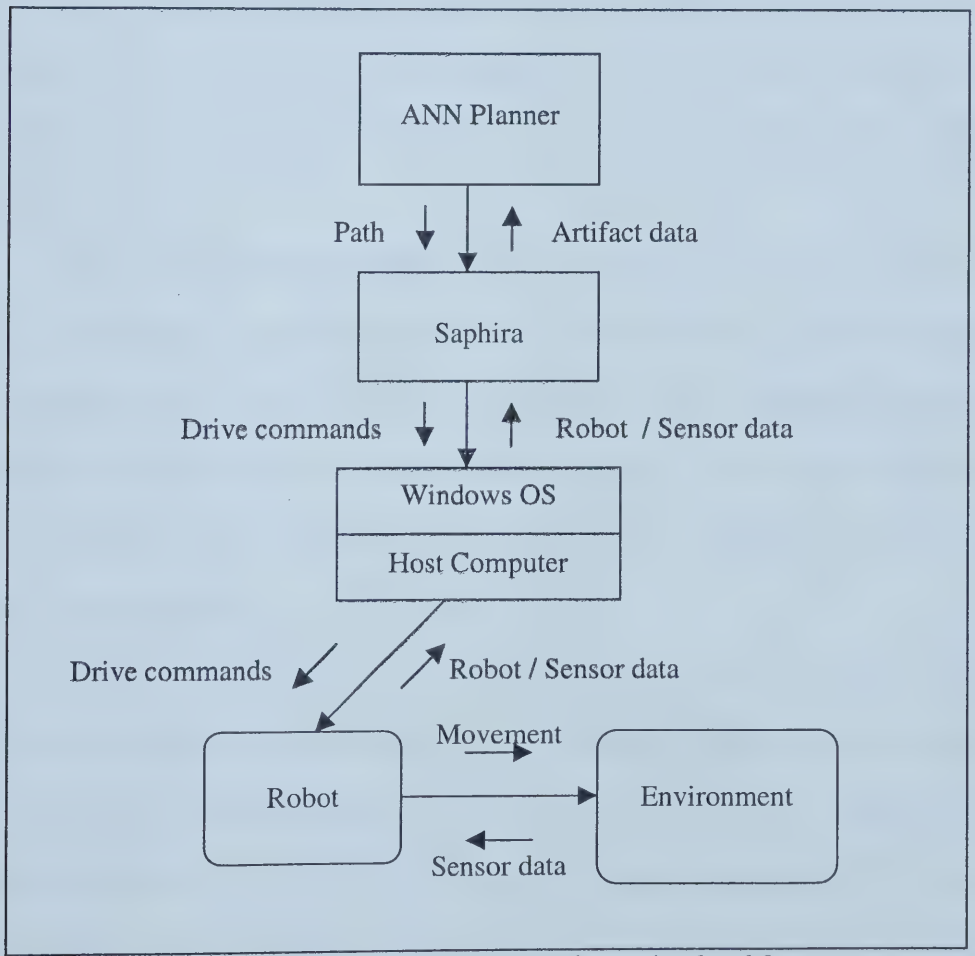


Figure 5.1: Complete System at abstraction level 0

Saphira for execution by the behaviour-based control architecture.

Saphira acts as the interface for the ANN motion planner to both the robot and the host computer for all operations. All commands and SIP's must go through the computer's Windows operating system between the Saphira and robot. The robot, either in the form of the simulator application or the physical robot, must interface with the host computer for command and data communication. This can take the form of RS-232 null modem cable, Ethernet connection, or the simulator running on the same computer. The robot interacts with the environment for command execution and retrieves raw sensor data and robot status data for transmission to Saphira via the communication link.

The ANN Planner requires updated artifact data about the presence of obstacles in the grid squares representing the neural map. The ANN Planner updates this neural map and then returns a path when one is completed or until a complete computation cycle has concluded that no path exists. The ANN Planner computation process must work with control tasks of the Saphira system under a finite state machine paradigm in order to run concurrently.

Figure 5.2 depicts a level 1 abstraction of the ANN motion planner. In this abstraction, the isolated planner incorporates all subcomponents needed to instantiate a 2D neural plane and update the path over time. The individual neurons were determined to be a class of individual elements in order to increase the flexibility in employment. As a simple array, these neurons would not be able to take on unusual shapes that would correspond to typical building floor plans without a great deal of

wasted data space.

Simple array storage would increase computation time by having a larger array than necessary for the path determination. As individual neurons, they can be created and linked into any arbitrary pattern that corresponds to the building floor plan.

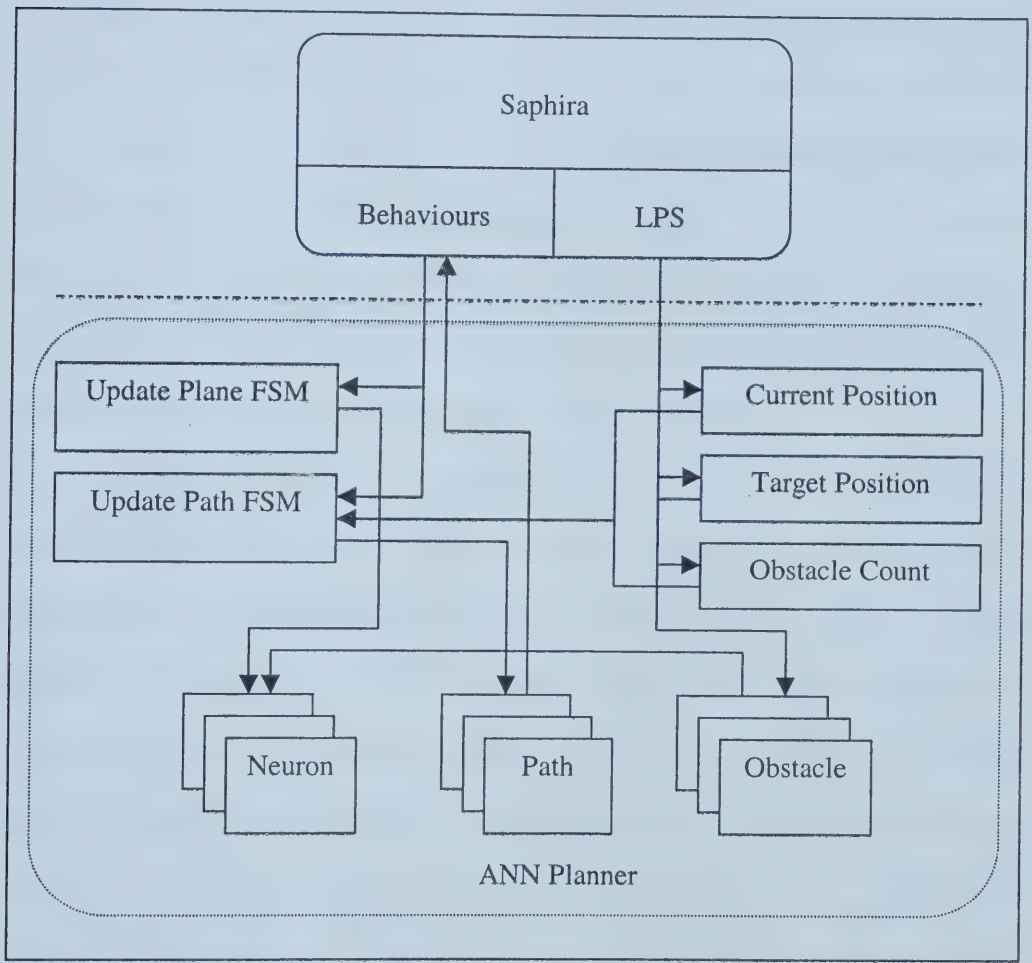


Figure 5.2: ANN Planner at abstraction level 1

The ANN motion planner receives data from Saphira's LPS and incorporates the presence of obstacles into the neural map. New obstacles or a change in the target or current poses is relayed directly to the ANN motion planner at the time of the

change. All other Saphira functions and micro tasks are hidden from the ANN planner.

The ANN Planner uses two finite state machines (FSM) to supervise the two major systolic processes Update Plane and Update Path. Update Plane executes neuronal modeling using the finite difference approximation over 10 states. Update Plane operates at 10 Hz within the drive motor update cycle. Update Plane takes about 2 seconds to complete an entire computation cycle if needed. Update Path uses a FSM to seek a path after every Update Plane cycle. Update Path transmits the final path to the behaviour shell once computed and triggers the Saphira behaviour shell to read it. Once a path exists, the Update Plane ceases computation and monitors changes in the target position, current position, or obstacles. If any of these variables changes, it resets the computation state of Update Plane to begin propagation.

The path class was determined necessary as a storage class of the final path as determined by the ANN Planner. Saphira has access to the path array, which is a list of configurations as intermediate poses from the current to the target pose. Saphira transforms the path list into an array of artifacts within the LPS. These artifacts allow the Saphira system to employ the path as any other object in the workspace. Behaviours then employ the path list to complete navigation. These behaviours will be outlined in the following section. Saphira uses the behaviour shell to trigger a state change of Update Plane and Update Path during the drive motor update cycle. The behaviour shell can also reset the Update Plane and Update Path FSM's when the behaviours are initialized or restarted with a new goal.

Object Classes

From the above breakdown of the ANN Planner, the following data classes were determined. These data classes are Coords, Obstacle, Path, CNeuron, and CPlane. Coords, Obstacle and Path are simple structures. CNeuron and CPlane are classes and are named in accordance with the Microsoft Foundation Class (MFC) naming convention.

Struct Coords

The path, obstacle and neural map data structures need a more basic data type that indicates an x and y coordinate in a form that can be used as a single value. Coords is a struct that represents the x and y Cartesian coordinates for a neuron in the plane and is represented in Figure 5.3. The 16 bits of x_coord is bit-shifted 16 bits into the most significant bytes of a 4 byte word that is the hash key of the neuron. The 16 bits of the y_coord remains in the least significant bytes. As unsigned short integers, the x and y coords are limited to 65536 distinct values. The hashed coordinates are used to link neighbouring neurons during intialization and for inserting and removing obstacles.

```
typedef  struct Coords {
        unsigned short int  x_coord;
        unsigned short int y_coord;
};
```

Figure 5.3: Struct Coords

Struct Path

Path is the culmination of the entire robot motion planning method. This structure is an array that contains a list of the neurons that must be traversed to reach the goal pose.

This array contains the x and y coordinates as well as an intermediate angle that aligns the robot for motion between intermediate poses along the path. These poses form the intermediate waypoints for motion from the current pose to the final pose.

Struct Obstacle

Obstacles is an array that represents those areas of the workspace that are occupied and cannot be traversed. This list contains the x and y coordinates for each region representing a grid square of Saphira's local perceptual space. These regions correspond to a neuron within the neural plane. This is a FILO list of obstacles that is updated as the microtasks within Saphira determine.

Class CNeuron

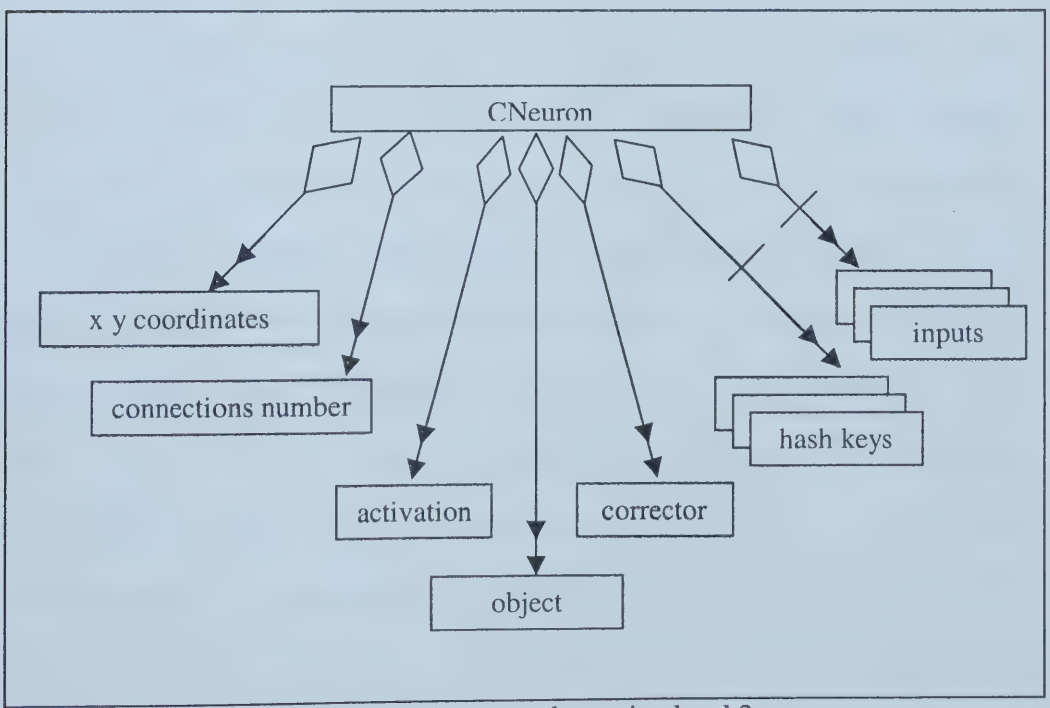


Figure 5.4: Neuron at abstraction level 2

Figure 5.4 depicts the neuron class of elements that make up the neural map in the ANN motion planner. The neurons contains 7 data types. The x and y coordinates determine the Cartesian grid number of the neuron. The number of connections indicates the number of neurons neighbouring that provide input and sets out the size of the inputs and hash keys arrays. The array of inputs are pointers to the neighbouring neurons' activation. The hash keys represent the unique number of the neighbouring neurons. The hash key values are hashed from the unique x and y coordinates of each neuron. The corrector is used for the predictor-corrector finite difference equations as an adjuster to the activation. The object indicates whether or not the neuron is occupied by an obstacle, a target, or free space. The object alters the dynamics of the finite difference equations as described in Chapter 3.

The class CNeuron uses many functions to operate the neurons within the neural map. CNeuron is the class constructor instantiates a neuron and inserts a desired x and y coordinate and an activation and initializes all other variables. ComputeNeuron_ExplicitEuler applies Equation (3.10) as a finite difference equation based on the 1st Explicit Euler method. ComputeNeuron_ImplicitEuler applies Equation (3.10) as a finite difference equation based on the 1st Implicit Euler method. ComputeNeuron_RungeKutta applies Equation (3.10) as a finite difference equation based on the 2^{nd} order Runge-Kutta method. ComputeNeuron_BogackiShampine applies Equation (3.10) as a finite difference equation based on the 3^{rd} order Bogacki-Shampine method. ComputeNeuron_RungeKutta4 applies Equation (3.10) as a finite difference equation based on the 4^{th} order Runge-Kutta method. SetActivation inserts an arbitrary value into the activation of the specified neuron. GetActivation retrieves

the activation value in the specified neuron. MeasureError retrieves the error function of associated finite difference equations like the 3rd order Bogacki-Shampine method. InsertTarget inserts a target into the object data variable. InsertObstacle inserts an obstacle into the object data variable. ClearObject clears the object variable in the specified neuron. ResetActivation resets the activation in the specified neuron to zero. ReduceActivation decreases the activation in the by an arbitrary factor. This can be used to reduce the activation of the entire network. ReturnObject checks for the pre-existence of either an target or obstacle at the specified neuron and returns the value. NewtonInterpolate is used by implicit numerical methods to interpolate the neural activation at any time step.

Class CPlane

The final class needed is the container class that contains the sum of the rest of the process. The class CPlane represents the entire ANN Planner including the classes and structures described above and other ancillary data types that facilitate the operation of the neural map. It is called CPlane to indicate that it represents a Cartesian plane of neurons.

Figure 5.5 indicates the data types that are contained in the CPlane class. The corner_neuron is a double precision float representing a boundary to the workspace. Connections for neurons at the edges point to this blank. The CMap class is a Microsoft Foundation Class template that allows a single unique value to represent the storage of a single object in a linked mapping of objects. The CMap class requires the

hashed value of the x and y coordinates to designate each object uniquely.

```
typedef struct CPlane
{
        double *        corner_neuron;          // the connection to a blank node
        double          blank;                  // an empty double
        class CMap( Neuron nodemap[MAX_NODES_ROW*MAX_NODES_ROW]);
        int     finalpathpoint;         // highlights the final path point in the path
        int     ObjectCounter;          // tracks the number of obstacles
        int     ANNState;               // state counter for FSM UpdatePlane
        int     FailureCount;           // indicates failure of  UpdatePlane FSM
        int     NoPath;                 // indicates a path exists
        int     FSMiterations;          // UpdatePlane state
        unsigned int    neuralcomputations; // for evaluation of FDE methods
        Coords          target;         // the neuron to move to
        Coords          current;        // the current location
        Coords          pasttarget;     // observes the previous location
        Coords          path[PATH_MULTIPLE*MAX_NODES_ROW];
        Coords          ObjectList[MAX_NODES_ROW*MAX_NODES_ROW];
        float           angle[PATH_MULTIPLE*MAX_NODES_ROW];
};
```

Figure 5.5: Class CPlane data values

The following functions are used to operate the neural map and update the path for use by Saphira. CPlane is the class constructor and it is responsible for instantiating the base and other classes using the passed variables. The neural map creates a single neuron and clones it to the shape of the workspace. ConnectPlane is used by the class constructor to link the neurons once they have been instantiated. This function creates arrays of hash keys and inputs only as large as the specific neuron requires to speed up the computation process. EncodePosition hashs two unsigned short ints into the unique neuron ID in an unsigned long int. Hashing bit-shifts the code_x into the most significant bytes of the ID and stores the code_y in the least significant bytes. DecodePosition de-hashs the unique ID unsigned long int and returns two shorts in the form of a Coords structure. PlaneInsertCurrent stores the x and y

coordinates in the Current Coord structure. PlaneInsertTarget function stores the x and y coordinates in the target Coord structure. PlaneInsertObstacle places an obstacle into the neuron at the specified x and y coordinates. A new obstacle is placed in the obstacle array. PlaneReplaceObstacle inserts a previously removed obstacle from the neuron at the specified x and y coordinates without altering the obstacle array. PlaneRemoveObstacle removes the obstacle from the neuron at the specified x and y coordinates but it does not remove the obstacle from the obstacle array. PlaneClearNode clears the activation, corrector, and object variables of the specified neuron. PlaneNewtonInterpolate commands the neuron at the specified x and y coordinates to interpolate activation using Newton's Method.

UpdatePlane is a finite state machine (FSM) that operates the neural map to determine the motion path from current pose to final pose. The finite state machine is described in Figure 5.6. The UpdatePlane FSM is iterated by the Saphira behaviour shell. The results of initial testing revealed that the process can be streamlined to speed up the solution of the motion path if the individual tasks of propagation and computation were separated. This process was divided into different states where different time steps and FDE's of varying accuracies were employed to arrive at the solution. The initialization state sets up the computing process. The reset state resets the neurons' activations for a faster computation of the path. In the propagation stage, the neural map neural activities are reset and the obstacles removed to allow propagation of an approximate result to all areas of the network. The time step for this step is several orders of magnitude smaller than the most stable time step and so the local and global errors are limited. During propagation the solution is advanced in a

radial pattern from the target neuron over a few iterations. Following the propagation, the obstacles are replaced and the neural map is ready for a more accurate solution using a larger time step. The computation step moves across the neural map over more iterations. At every iteration of the Computation state, the node map is searched to determine if a solution exists. If a solution is found, UpdatePlane moves to the Success State and refines the solution. If success is not reached in the allotted number of iterations, then the Failure state is reached which adjusts the FDE methods to allow for a more precise computation.

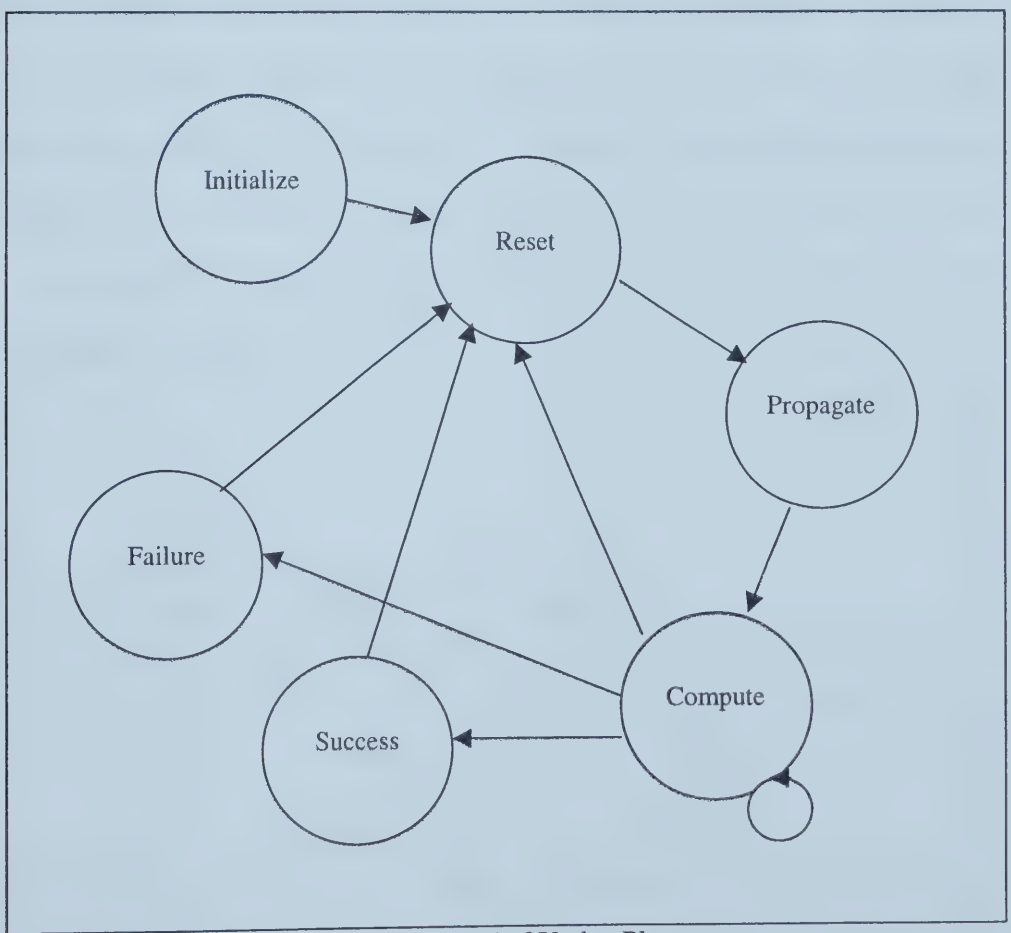


Figure 5.6: FSM of UpdatePlane

Update Path is a finite state machine that searches the neural map for the motion path and stores this in the path array. Figure 5.7 illustrates the states of Update Path. The UpdatePath FSM is iterated by the Saphira behaviour shell. UpdatePath converts the neural map coordinates for each neuron into artifacts centred on each grid square using equation (5.1)

$$x = (GRIDSIZE * y) + (\frac{GRIDSIZE}{2})$$
$$y = -(GRIDSIZE * x) + (\frac{GRIDSIZE}{2}) \quad (5.1)$$

where GRIDSIZE is the dimension of the area covered by the neuron. Note that the Neural map is aligned with the traditional Cartesian plane and requires a conversion of axes to correspond to the workspace {0}. UpdatePath then iterates through the path array and determines an angle from waypoint (i-1) to waypoint (i) that aligns the robot for smooth motions between points. Initialize starts the process of updating the path as the finite state machines are initialized.

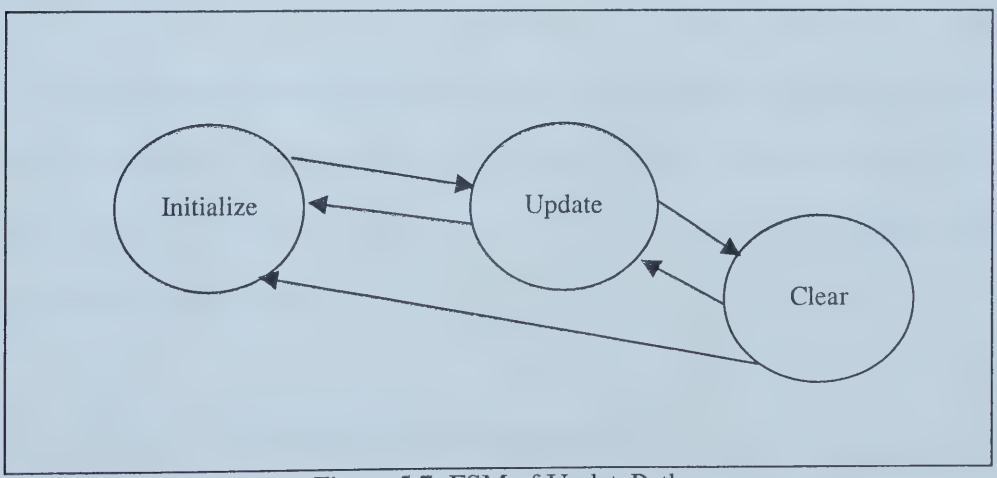


Figure 5.7: FSM of UpdatePath

The Update state reads the ANN Planner path and places it into an artifact point list

with other Saphira artifacts for use by the control architecture. The Clear state is then used to remove some of the unused artifacts from the previous path for clarity. At any time, the FSM may be reset to restart the entire ANN Planner.

ProcessPlane_Propagation uses the PlaneRemoveObstacle and PlaneReplaceObstacle to propagate the neural activity across neural map. This function uses a radial pattern to iterate through the network and bring the various regions into an approximate solution prior to completion by ProcessPlane_Computation. This function is called by the Update Plane FSM to reset the network prior to calling ProcessPlane_Computation.

ProcessPlane_Computation iterates through the network to resolve the final path. It uses the most accurate finite difference equations (4^{th} order Runge-Kutta) with a stable time step. ProcessPlane_Refinement is used to improve the resolution of the neural map while the robot is moving. The FDE time step is significantly smaller than for ProcessPlane_Computation. ProcessPlane_ResetPlane iterates through the entire neural map and resets the activations of each neuron to zero. ProcessPlane_ReducePlane reduces the neural activations of all neurons in the neural map by an arbitrary factor. SeekPath starts at the current Coords and searches the neural map by the gradient ascent rule to find the target Coords. PlaneResetPath function resets the path array.

5.2 Hybrid Architecture Implementation

This section describes the interface between the deliberative ANN robot motion

planner and reactive behaviour-based control architecture. The behaviour sfTotalPath is a new type of behaviour that does not compete for control of the robot motion commands but uses a FSM to control the operation of other reactive behaviours. This behaviour is supported by the behaviours sfGoToPos and sfRotateToAngle that execute basic motion. The use of these three behaviours allows the Pioneer robot to achieve complex deliberate motions while still allowing reactive behaviours to avoid collisions. These behaviours are used in conjunction with pre-existing Saphira avoid collision behaviours to serve as the hybrid robot control architecture.

sfTotalPath

This behaviour uses the path array from the current pose to the final pose indicated as a set of intermediate poses to be moved through by the robot. Since the Pioneer has a turn radius of zero, it is possible to achieve any arbitrary orientation by simply rotating. sfTotalPath decomposes the entire motion into a series of translations and rotations that are carried out by the behaviours sfGoToPos and sfRotateToAngle. sfTotalPath sets the first neuron as the waypoint for motion and then turns on and off the other behaviours as the situation dictates to achieve that waypoint. Once this point is achieved, sfTotalPath sets the next point as the waypoint and continues until the final neural waypoint is reached. Once the last neural waypoint is achieved then sfTotalPath sets the arbitrary final pose (which may not be the centre of the region) and completes the final motion. This behaviour is given the second lowest priority so avoid behaviours can react to obstacles.

sfGoToPos

This is a behaviour is used to move to an arbitrary position from the current position of the robot. sfGoToPos has been modified from the original Saphira version to use a waypoint in order to arrive at the desired position aligned with the desired orientation at that point. This ability decreases the time needed to rotate by sfRotateToAngle to achieve the desired angle. sfGotToPos uses both translational velocity and rotational velocity setpoints to move towards the desired pose. sfGoToPos uses a fuzzy desirability function to determine how near the current robot pose is to the desired pose. As the robot approaches the desired pose, the goal state approaches unity. The behaviour turns itself off once the robot reaches an arbitrary radius of the desired pose. sfGoToPos is given a higher priority than sfTotalPath but a lower priority than the avoid behaviours.

sfRotateToAngle

sfRotateToAngle holds the translational velocity at zero altering the rotational velocity to align the robot with the desired orientation. The difference between the current angle and the desired angle is measured as a fuzzy desirability function (as described in Saffiotti *et. al.* 1999) and impacts the direction and the magnitude of rotation. The goal state approaches unity as the current angle approaches the desired angle. The behaviour turns itself off once it reaches an arbitrary difference with the desired angle. sfRotateToAngle is given the same priority as sfGoToPos and a lower priority in

relation to the avoid behaviours. sfRotateToAngle attempts to compensate for the latency in response and the inertia of the robot by braking rotations prior to reaching the desired angle.

Avoid Behaviours

sfTotalPath, sfGoToPos, and sfRotateToAngle are integrated into the Saphira behaviour shell with the other existing behaviours. These other behaviours control the robot in cases where the LPS indicates that a collision is imminent. These avoid behaviours are sfAvoidCollision, sfAvoidRearCollision, sfStopCollision. sfAvoidCollision, and sfStop. sfAvoidCollision monitors the LPS ahead, left, and right of the robot and triggers if objects approach too near the robot. sfAvoidRearCollision monitors the LPS behind of the robot and triggers if objects approach too near the robot. sfStopCollision monitors the LPS ahead of the robot and triggers if objects approach too near the robot by slowing to zero translation velocity. These behaviours are given the highest priority in the behaviour hierarchy and will subsume all other lower behaviours in the case of imminent collision. If the conditions do not indicate collisions then these behaviours are not triggered and the lower priority navigation can be conducted. sfStop is safety behaviour that alters the translational velocity setpoint to zero. sfStop is assigned the lowest priority, so that in the absence of higher priority objectives the robot will remained stopped. Figure 5.8 illustrates the behaviour hierarchy of the Saphira control architecture. The lowest priority is the sfStop behaviour and the highest are the avoid collision behaviours. Under the Saphira

behaviour resolution scheme, the activations of the same priority behaviours are averaged using a centre of mass to resolve the final drive commands and the other priority behaviours are ignored.

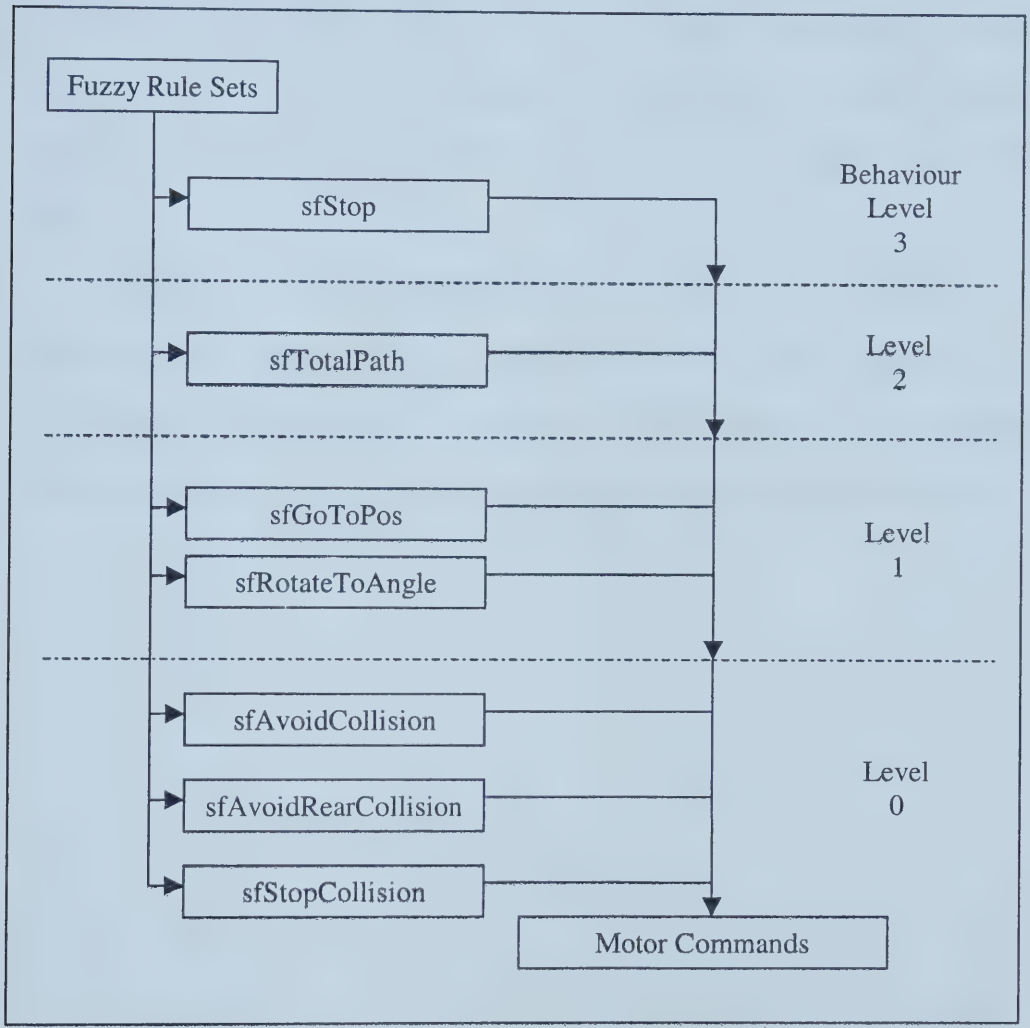


Figure 5.8: Saphira behaviour hierarchy

5.3 Software Metrics

The ANN Planner required development using a spiral-engineering model using design, testing, and evaluation iteratively to improve the overall performance. The

CNeuron required 3 versions to reach the final structure. CPlane required 2 versions to determine the optimal amount of ancillary variables. The ConnectPlane and SeekPath functions required 3 iterations to arrive at the fastest method. ProcessPlane_Computation needed the most improvement, undergoing 8 variation algorithms until the speed of computation was sufficient for real-time operation. Overall, the code for the ANN Planner required approximately 10000 lines of code (LOC).

Work on the Saphira behaviour-based architecture required significant time testing the functions and behaviours that were part of the original application. The three designed behaviours were implemented in C in order to fit into Saphira. Modifications to the robot control architecture required approximately 2000 LOC.

Chapter 6

Experimental Results

This chapter describes the results obtained during the testing and evaluation of the ANN robot motion planner operating on the Pioneer robot. The first section describes the evaluation of various numerical methods that were tested for the finite difference equations used to conduct the neuronal modeling. The second section presents the results obtained using the ANN planner for various extensions of the advanced motion problem.

6.1 Numerical Methods Results

Evaluation of the various finite difference approximation methods resulted in the determination of the optimal values of the coefficients of the neuronal models working in this application. Based on a double precision (64-bit) floating point variable for the neuron activations, the values in Table 6.1 were determined the best for modeling.

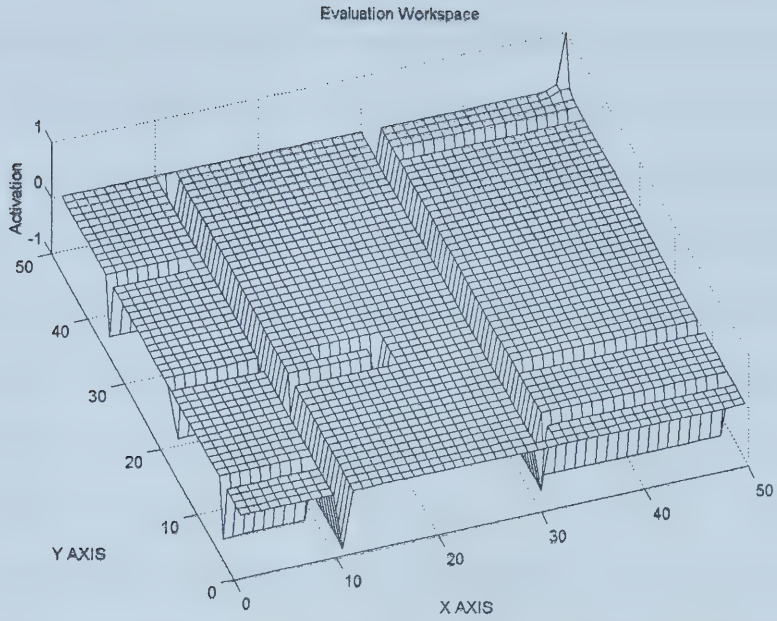


Figure 6.1: Evaluation workspace for FDE methods

Table 6.1: Neuronal Model Coefficients

A	11.000
B	1.000
D	1.000
g	1.000
H	100.000

The optimal FDE for the ANN Planner was determined by using a moderately occupied test workspace to compare the results of each FDE. The 50 by 50 neural map ANNTest, as depicted in Figure 6.1, was used as the evaluation workspace. A 50 by

50 map was chosen to make the resulting network comparable to the Hopfield network in Lagoudakis 1996. This evaluation workspace provides a reasonable case for static obstacles. If the evaluation workspace approaches a maze-like case where the obstacles occupy a majority of the neurons then the solution is determined as quickly as in the case where the workspace is close to empty. The length of the ideal path in the evaluation workspace ANNTest was determined to be 209 neurons in length from the (0,0) neuron in the lower left corner to the (49,49) neuron in the upper right corner. The lower bound activations of –1.0 represent neurons indicating obstacles present. The single peak of +1.0 indicates the target position.

Each FDE method computed the ideal path within the ANNTest workspace for a specified number of iterations and then the neural map was searched to determine if the path existed from the current to target neurons. This process was repeated until the path was found. Several different iterations counts were tested until the fastest solution for each FDE was obtained. The optimal results are recorded in Table 6.2.

The neural computations indicate how many times the individual neurons were time-marched forward. The total computations column indicates the total number of equations needed to complete finite difference approximation. The neural and total computations for the implicit Euler method do not include the interpolation of the activation at time step t+1, ξ_{t+1}, which required Newton's backward difference method a variable number of interpolations for each neuron iteration. During evaluation of different FDEs, it became apparent that the use of various time steps resulted in marked differences in the accuracy of the estimation. This was noted in Hoffman 1992 as an indication of a stiff ODE. For very small time steps (on the order of 1/1000000s), the

low order FDE equations would reach dynamics similar to the higher order equations

Table 6.2: Optimal results for various FDE methods against evaluation workspace.

Method	Order	Time (s)	Neural Computations	Total Computations	Global Error
Euler Explicit	1^{st}	~3	355047	355047	$O(\Delta t)$
Euler Implicit	1^{st}	~3	>355047	>355047	$O(\Delta t)$
Euler Predictor-Corrector	2^{nd} (1)	~2	355047	710094	$O(\Delta t^2)$
Heun's Modified	2^{nd} (1)	~3	355047	710094	$O(\Delta t^2)$
Runge-Kutta Midpoint Predictor-Corrector	2^{nd}	~2	355047	710094	$O(\Delta t^2)$
Bogacki-Shampine	3^{rd} (2)	~3	285047	855141	$O(\Delta t^3)$
Runge-Kutta	4^{th}	~3	285047	1140188	$O(\Delta t^4)$

with much larger time steps. The number of iterations needed to compute a solution with low order FDEs approached the number of computations carried out by the higher order FDEs and so no computational improvement was realised. At the same time, the higher order FDEs require more computations per iteration and the value of these computations is lost when the neural activity is still propagating outward from the target neuron because a significant portion of the neural map still has zero activation and so the net effect of the computation is negligible at the areas farthest away from the target. This led to the realisation that the Grossberg shunting equation ODE was a stiff equation for FDE methods, as defined by Hoffman 1992, because the accuracy in

activations varied across the scale of time steps. These observations led to the adoption of a modified computation process that separates the propagation of the neural activation from the accurate computation of the final path. The 2nd order Runge-Kutta was chosen to propagate the neural activation at a very small time step (on the order of 1/1000000th the time step of the higher order FDE) because it was fast with reasonable global error at small time steps. The second

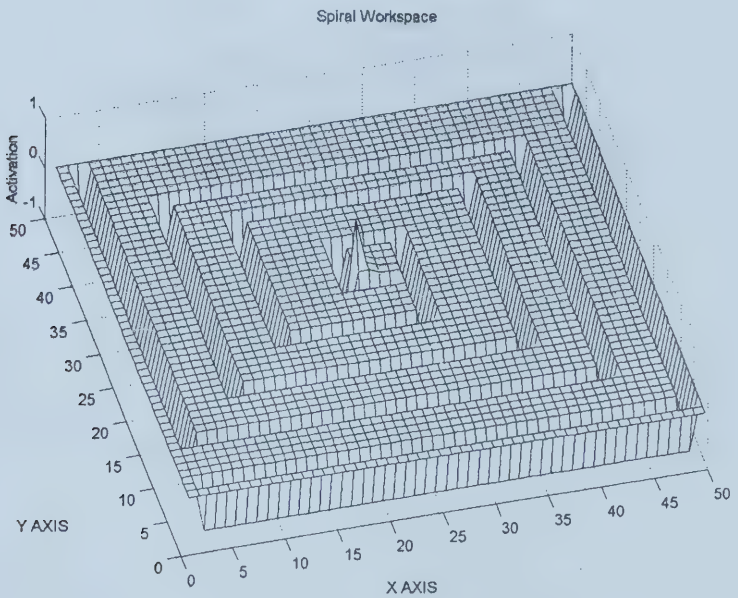


Figure 6.2: Spiral workspace final activation graph

phase of the computation was conducted by the 4th order Runge-Kutta to compute the final neural activation because of the low global error. This modified method brought the overall computation time less than 2 seconds within the finite state machine process. The following figures describe the final activation graphs of the ANN planner compared with some commonly used examples workspaces. Figure 6.2 shows the target neuron in the middle of the neural map. The free space neurons' activations are

very small numbers approaching zero proportional to their distance from the target neurons. Figure 6.3 shows the final activations for a workspace for Hopfield neural networks in Lagoudakis 1996.

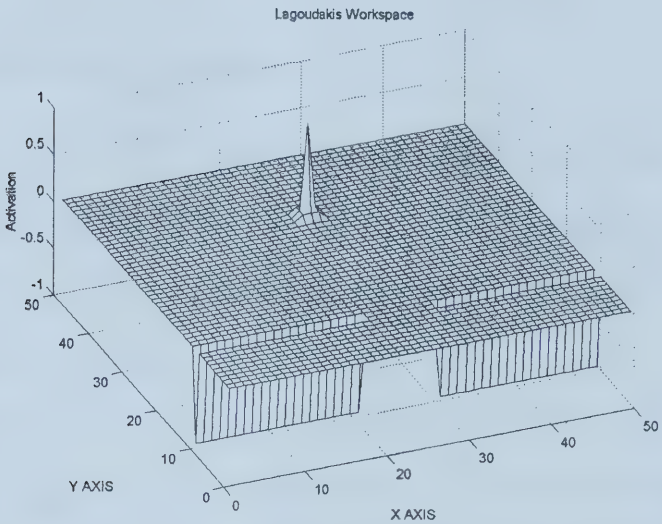


Figure 6.3: Lagoudakis simple workspace final activation graph

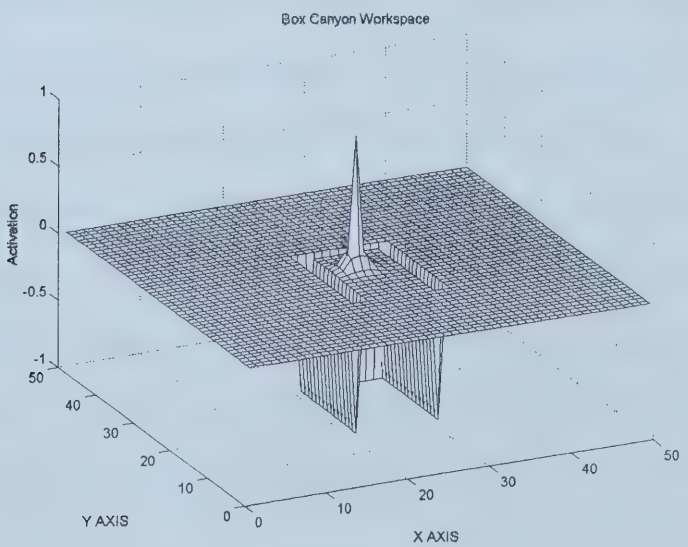


Figure 6.4: Box canyon workspace final activation graph

Figure 6.4 illustrates the activation levels for a target neuron surrounded by a box canyon. Unlike the potential field method, the non-learning ANN network based on the Grossberg shunting equation does not become "stuck" in a potential minima near an obstacle like the box canyon.

6.2 Advanced Motion Results

The ANN planner was tested against the ANN test workspace as shown in Figure 6.5 using the Saphira simulator. The robot was connected to Saphira and allowed to randomly pick poses within the room and move from its current position to the target pose. This process continued until a large enough sample was gathered. The position could be any arbitrary x and y coordinates and did not have to end on the centre of the neural grid. The angle was randomly determined as a multiple of 10 degrees in the interval (0,360]. The robot was given a translational velocity of 300 mm/sec as the upper bound for all motions and a 39 degrees/sec upper bound for rotational velocity.

The robot was given ideal location information by turning off the encoder jitter, angle jitter and angle drift that occur from the drive wheel encoders. This testing involved the advanced motion with kinematic constraint and uncertainty in obstacle location extensions. The uncertainty in obstacle location is a result of the sonar sensors returning a rough location for obstacles and the characteristic specular reflections that occur with sonar. The variant parameter for this testing was the absolute radius achieved by the robot near the target location, where the robot

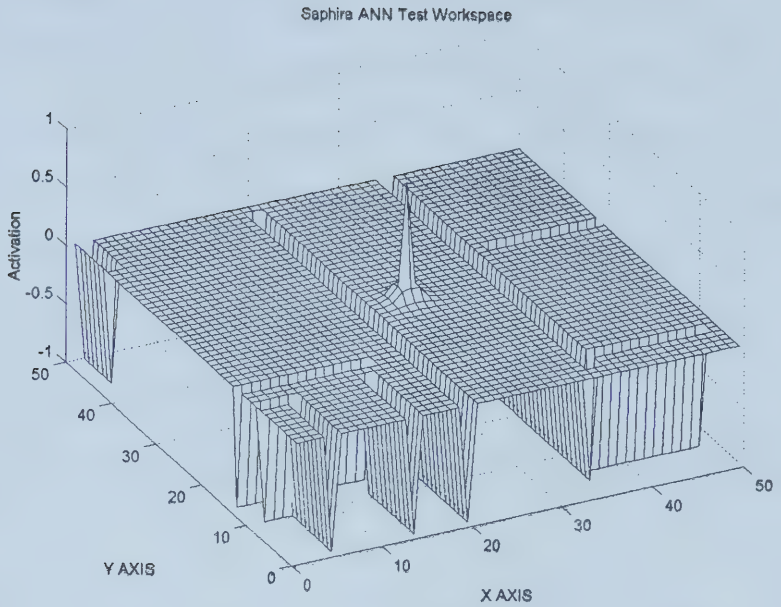


Figure 6.5: Saphira test workspace ANNTest

determines it has successfully completed the desired motion when its Cartesian x- or y-coordinate is within the specified radius of the desired position. The two values used were 30mm and 35mm respectively. The actual values of x, y, and θ were compared with the desired values to determine Δx, Δy, and $\Delta\theta$. Table 6.3 describes the results for 30mm with a sample population of 1200.

Table 6.3: Results of Advanced Test #1 for Positional Radius 0f 30mm

	Δx (mm)	Δy (mm)	ΔΘ (degrees)	Time (s)	Path Length (# of neurons)	ANN Computations (iterations)
Average	7.14	7.17	0.76	140.19	29.45	4.25
Median	1.83	4.28	-0.01	91.89	19.00	4.00
Std. Dev.	83.19	29.62	7.63	130.99	29.43	0.88

Table 6.4 describes the results for a 35 mm position radius with a sample population of 1304.

Table 6.4: Results of Advanced Test #1 for Positional Radius 0f 35mm

	Δx (mm)	Δy (mm)	ΔΘ (degrees)	Time (s)	Path Length (# of neurons)	ANN Computations (iterations)
Average	5.60	18.67	0.00	129.46	30.15	4.05
Median	4.87	16.39	0.02	89.66	19.00	4.00
Std. Dev.	24.92	35.89	1.34	102.12	26.24	0.37

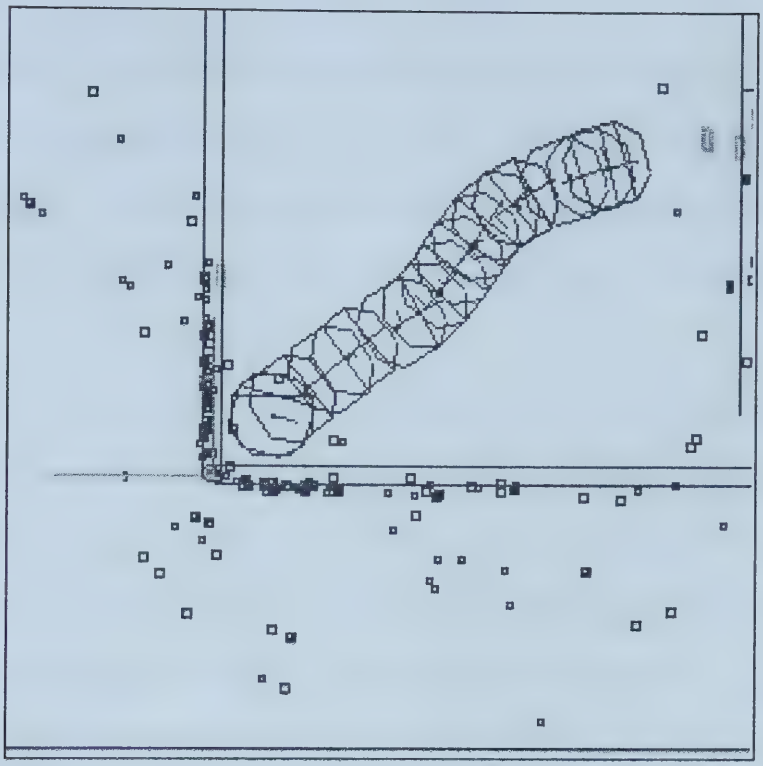


Figure 6.6: Motion of the Pioneer 2 DX simulator within Saphira showing sonar locations and wall artifacts

The random movement of the robot generated a largest path of 163 neurons and a smallest path of 2 neurons. The overall number for computations remained constant at 4 iterations without a great deal of variation for the size of the path. The attempt to reach a radius of 30mm from the target position resulted in longer times of motion and a larger variation in the x and y accuracies.

The average time to traverse one neural square, at most a distance of 1.06m from corner to opposing corner, was 4.29 seconds. The shorter paths (those below the statistical mean) had a longer average time and those above the statistical median had a shorter average time per neuron. This was computed by adding all the path lengths of

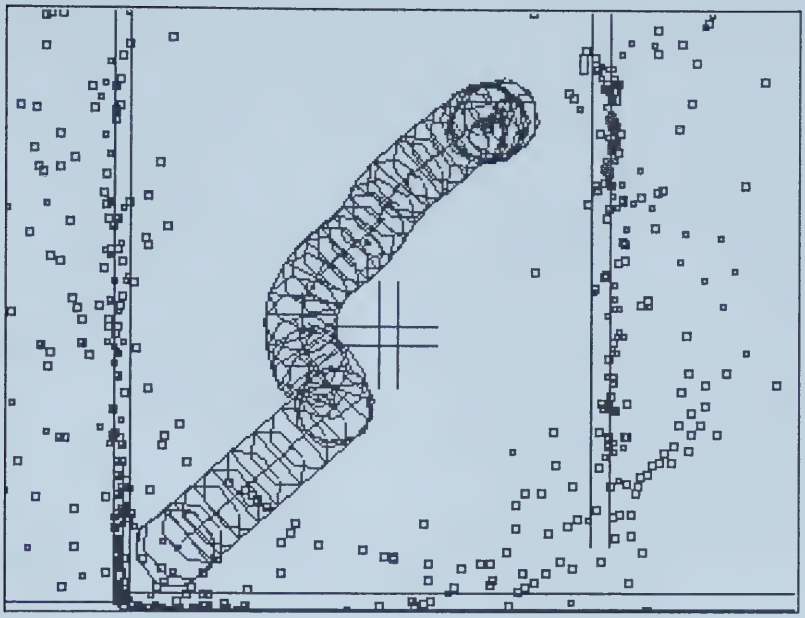
all motions and dividing by the total number of neurons.

The behaviour sfRotateToAngle proved able to approach the desired angle with reasonable accuracy, although there were spurious results of over 20 degree differences in the attempts to reach the 30mm positional radius. This was due in part to the avoid collision behaviours that counteracted the desired rotation when the goal pose was near a wall.

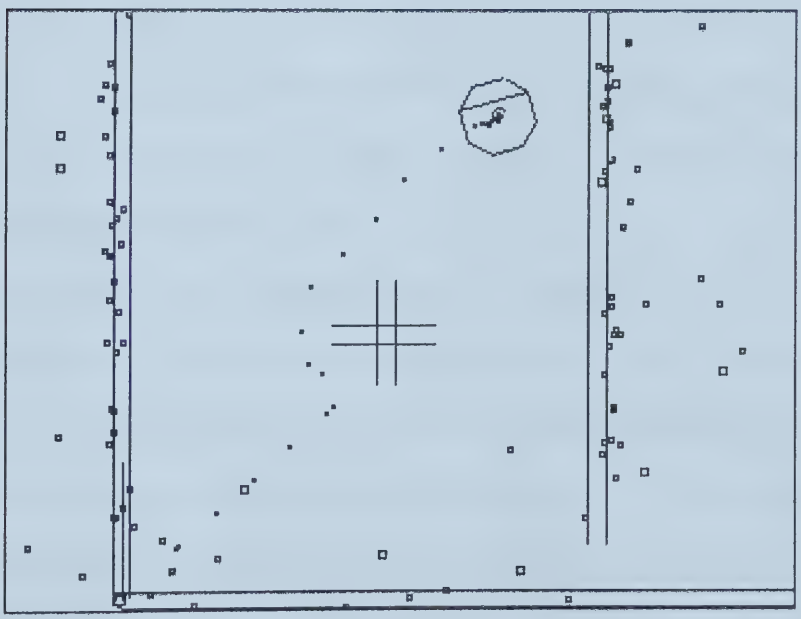
The robot simulator underwent motions with obstacles inserted during movement so the ability to handle a changing environment could be observed. Figure 6.7 describes the motion from pose (375.0, 375.0, 0) to (3433.3, 2683.3, 20) which is within neuron [4,3]. This was obtained by running Saphira in single step mode and a composite image was developed over the course of movement. Panel (a) describes the composite motion, while Panel (b) shows the last pose of the robot when it reached the final pose of (3428.2, -2662.9, 19.95) and the sfTotalPath behaviour turned off all motion behaviours ceased all further movement. An obstacle was inserted at neuron [2,2] corresponding to position (1875.0, -1875.0) when the robot was within the region of neuron [1,1] at (1125.0, 1125.0). The avoid collision behaviours responded by slowing down forward motion and initiated a turn to its left (positive rotation) to avoid the obstacle within region [2,2]. The UpdatePath FSM resolved a new path within 2 single step iterations. Once the robot was facing 90 degrees away from the obstacle, it began forward motion and continued to move around the obstacle. Once the collision behaviours subsided, the robot resumed course for the final pose. The difference between the final pose and the desired pose are listed in Table 6.5.

Table 6.5: Motion differences for movement

Δx	-5.167mm
Δy	+20.440mm
$\Delta \theta$	-0.05 degrees



(a)



(b)

Figure 6.7: Motion of the robot under transient obstacle.

Panel (a) Composite motion, Panel (b) Final pose and wake from motion

Chapter 7

Summary

The non-learning ANN robot motion planning method was demonstrated on the Pioneer 2 DX. It was shown that the ANN Planner is capable of reaching the goal pose within the specified positional radius as described in Chapter 6. This chapter provides some discussion of the results.

The shunting equation used for neuronal modeling was shown to be a stiff ODE, as defined by Hoffman, due to the varied accuracy of the neuronal modeling across time steps. The stiffness of the ODE was further complicated by the need to propagate the solution of the motion path to all corners of the neural map without *a priori* knowledge of the obstacles and the current location. An attempt was made to use low order FDE methods to increase the number of complete neural map computations that could be made in real-time in order to fit within the Saphira architecture. Implicit, explicit and predictor-corrector methods were tested, and the trade-offs of instability of lower order explicit FDE methods versus more computations with higher order FDE methods were examined. Experiments with a smaller time step

approach versus longer time step with lower order methods did not significantly alter the time to find solution. This led to a modified method. The modified method uses a low order FDE at a small time step to propagate the solution and a higher order FDE to compute a real solution. The increased the speed of computation was observed to provide a solution in all occasions in time for the motion of the robot. This improvement is due to the fact that the solution is only a tool and therefore the actual activation values may not be close to the approximated value provided by the FDE method. As long as there exists a difference in activation values from node to node then the gradient search will find the same path between neurons.

During testing on the Saphira simulator, the robot was shown to be capable of moving an arbitrary distance and achieve arbitrary poses when given ideal sensor information. The robot control architecture was able to react to the imminent threat of collision and once that threat subsided then the deliberative ANN motion planner again reasserted itself and headed the robot towards the goal pose. This method has shown that it can react to changing conditions.

One advantage this method has is that transient obstacle can be inserted or removed from the ANN Planner without needing to restart the computation process. When an obstacle is inserted, the neural activation is now negative and therefore the subsequent drive cycle path search will avoid this neuron even if it was on the path the iteration before. If an obstacle is removed, it will rapidly be adjusted for by the neighbouring neurons with a positive activation. The one condition for restarting the path computation is when the target neuron is moved. This is due to the fact that the previous target location has remained active and even though the activation in

neighbouring neurons is decaying it can cause temporary saddle points in the neural map that will hamper gradient search method. It was found that restarting with a zero activation and propagating out from the new target was faster than allowing the previous activations to remain.

In a realistic situation, the interplay between the ANN planner, data fusion, and localization will result in different accuracies and capabilities for the Pioneer 2 DX. Without global knowledge of obstacles and precise localization, then the resulting pose will occur within a larger region of the desired pose. This ANN Planner will work well with occupancy grid mapping models due to the grid-based nature the two methods share. Localization will also be more practical if the robot can be located within a grid square rather than a precise pose for every intermediate configuration. This will improve the in-transit planning and navigation of robotics by lowering the need for computational-complex localization methods. Once the robot reaches an arbitrary radius of the target location, then the complete localization methods may be employed to estimate the pose of the robot.

Bibliography

Ahuactzin J., El-Ghazali T., Bessiere P., Mazer E., Using Genetic Algorithmns for Robot Motion Planning, *10th European Conf. on Artificial Intelligence*, London, 1999.

Araujo F., Ribeiro B., Rodrigues L., A Neural Network for Shortest Path Computation, *University of Lisboa Report*, Lisboa, 2000.

Arkin R., Behavior-Based Robotics, MIT Press, 1998.

Baase S., Computer Algorithms: Introduction to Design and Analysis, Second Edition, Addison-Wesley, Reading, 1993.

Borenstein J., Real-time Obstacle Avoidance for Fast Mobile Robots, *IEEE Transactions on Systems, Man, and Cybernetics* Vol. 19, No. 5 Sept/Oct 1989, pp. 1179-1187.

Bouilly B., Simeon T., Alami R., A Numerical Technique for Planning Motion Strategies of a Mobile Robot in Presence of Uncertainty, *IEEE International Conference on Robotics and Automation*, Nagoya, 1995.

Brock O., Khatib O., High-Speed Navigation Using the Global Dynamic Window Approach, *Stanford University Report*, Stanford, 1998.

Brooks R., A Robust Layered Control System for a Mobile Robot, *A.I. Memo 864 Massachusetts Institute of Technology Artificial Intelligence Laboratory*, Boston, 1985.

Burlina P., DeMenthon, Davis L., Navigation with Uncertainty: Reaching a Goal in a High Risk Region, *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, Nice, 1992.

Canny J., Lin M., An Opportunistic Global Path Planner, *University of California - Berkeley Report* , 1992.

Canny J., The Complexity of Robot Motion Planning, MIT Press, 1988.

Cherif M., Laugier C., Milesi-Bellier C., Planning the Motions of an All-Terrain Vehicle by using Geometric and Physical Models, *IEEE Conference on Robotics and Automation*, San Diego, 1994.

Chen D., Szczeba R., Uhran J., Using Framed Quadtrees To Find Conditional Shortest Paths In An Unknown 2-D Environment, *Technical Report #95-2 University of Notre Dame*, 1995.

Connolly C., Burns J., Weiss R., Path Planning Using Laplace Equations, *IEEE International Conference on Robotics and Automation*, Nice, 1990, pp.2102-2106.

Cook D., A Hybrid Approach to Improving the Performance of Parallel Search, *University of Texas-Arlington Report*, 1995.

Dacre-Wright B., Laumond J., Alami R., Motion Planning for a Robot and a Movable Object amidst Polygonal Obstacles, *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, Nice, 1992.

D'Andrea-Novel B., Bastin G., Campion G., Dynamic Feedback of Nonholonomic Wheeled Mobile Robots, *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, Nice, 1992.

Elnagar A., Basu A., Heuristics for Local Path Planning, *Proceedings of the IEEE International Conference on Robotics and Automation*, Nice, 1992.

Fekete S., Houle M., Whitesides S., New Results on a Visibilty Representation of Graphs in 3D, *University of Cologne Report*, 1995.

Foskey M., Garber M., Lin M., Manocha D., A Voronoi-Based Hybrid Motion Planner for Rigid Bodies, *University of North Carolina-Chapel Hill*, 1998.

Fox D., Burgard W., Thrun S., A Hybrid Collision Avoidance Method For Mobile Robots, *Proc. of the IEEE International Conference on Robotics and Automation*, Leuven, 1998.

Glasius R., Komoda A., Gielen S., Neural network dynamics for path planning and obstacle avoidance, *Neural Networks* 8 (1), 1995, pp. 125-133.

Grossberg S., Contour enhancement, short term memory, and constancies in reverberating neural networks, *Studies in Applied Mathematics*, 52, 1973.

Goodrich M., O Dunlaing C., and Yap C., Constructing the Voronoi Diagram of a Set of Line Segments in Parallel, *Lecture Notes in Computer Science: 382, Algorithms and Data Structures*, Springer-Verlag, 1989.

Gurvitz L., Averaging Approach to Nonholonomic Motion Planning, *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, Nice, 1992.

Guzzioni D., Cheyer A., Julia L., and Konolige K., *Many Robots Make Short Work*, SRI International Report on 1996 AAAI Robot Contest, San Francisco, 1996.

Hsu D., Latombe J.-C., Motwani R., Path Planning in Expansive Configuration Spaces, Stanford University Report , 1996.

Hodgkin A., Huxley A., A quantitative description of membrane current and its application to conduction and excitation in nerve, *J. Physiol. Lond.* 117, 1952.

Hoffman J., Numerical Methods For Engineers and Scientists, McGraw Hill, New York, 1992.

Hopcraft J., Joseph D., Whitesides S., Movement Problems for two-dimensional linkages, *SIAM Journal Computing*, 13, 1984.

Hopcraft J., Wilfong G., Reducing multiple object motion planning to graph searching, *SIAM Journal of Computing*, 1986.

Hopcraft J., Schwartz J., Sharir M., Planning, Geometry, and Complexity of Robot Motion, Ablex Publishing Corporation, Norwood, 1987.

Horton I., Introduction to Visual C++ Standard Edition, Wrox Press, First Edition, Chicago, 1998.

Joseph D., Plantiga W., On the Complexity of Reachability and Motion Planning Questions, *Proceedings of 1st ACM Symposium on Computer Geometry*, 1985.

Junjie J., Baoquan L., Wei C., Jianing Y., Design Three-Dimensional Local Path Planner for AUV, *Proceedings of the 1999 IEEE Canadian Conference on Electrical*

and Computer Engineering, Edmonton, 1999.

Kavraki L., Latombe J.-C., Probabilistic Roadmaps for Robot Path Planning, J.Wiley & Sons Press, 1997.

Kernighan B., Ritchie D., The C Programming Language, Second Edition, Prentice Hall Press, Englewood Cliffs, 1988.

Khatib O., A Unified Approach For Motion and Force Control of Robot Manipulators: The Operational Space Formulation, *IEEE Journal of Robotics and Automation Vol. RA-3 No. 1 February 1987*,pp. 43-53.

Khatib O., Real-Time Obstacle Avoidance for Manipulators and Mobile Robots, *The International Journal of Robotics Research* Vol. 5 No. 1, Spring 1986, pp. 90-98.

Kim J., Amato N., Lee S., An Integrated Mobile Robot (Re)Path Planner and Localizer for Personal Robots, *Texas A&M University Report*, 1998.

Knobbe A., Overmars M., Kok J., Robot Motion Planning in Unknown Environments using Neural Networks, Utrecht, 1996.

Konolige K., Myers K., Ruspini E., The Saphira Architecture: A Design for Autonomy, *Journal of Experimental and Theoretical Artificial Intelligence JETAI Special Issue on*

Architectures for Physical Agents, 1997.

Konolige K., Myers K., The Saphira Architecture for Autonomous Robots, *SRI Report*, San Francisco, 1996.

Konolige K., Colbert: A Language for Reactive Control in Saphira, *SRI Report*, San Francisco, 1996.

Konolige K., Operation Manual for the Pioneer Mobile Robot, *SRI Report*, 1995.

Kortenkamp D., Huber M., Koss F., Belding W., Lee J., Wu A., Bidlack C, Rodgers S., Mobile Robot Exploration and Navigation of Indoor Spaces Using Sonar and Vision, *AIAA/NASA Conference on Intelligent Robots in Field, Factory, Service, and Space (CIRFFSS) 94*, University of Michigan, 1994.

Kortenkamp D., Bonasso R., and Murphy R., Artificial Intelligence and Mobile Robotics: Case Studies of Successful Robot Systems, AAAI/MIT Press, Menlo Park, 1998.

Krogh B., A Generalized Potential Approach to Obstacle Avoidance Control, *SME-RI Technical Paper MS84-484*, Society of Manufacturing Engineers, Dearborn, 1984.

Krose B., van Dam J., Neural Vehicles, *University of Amsterdam Report*, 1996.

Lagoudakis M., Maida A., Neural Maps for Mobile Robot Navigation, *Duke University Report*, 1998.

Lagoudakis M., Hopfield Neural Networks for Dynamic Path Planning and Obstacle Avoidance, University of Southern Louisianna, 1996.

Lambert A., Fraichard T., Landmark-based Safe Path Planning for Car-Like Robot Motion, Inria Rhone-Alpes, 2000.

Langer D., Rosenblatt J., Hebert M., A Behavior-Based System For Off-Road Navigation, Robotics Institute Report, 1993.

Latombe J-C. , Robot Motion Planning, Kluwer Academic Publishers, Boston, 1991.

Laumond J., Simeon T., Notes on Visibility Roadmaps and Path Planning, *LAAS-CNRS*, Toulouse, 1994.

Mandelbaum R., Mintz M., Active Sonar Fusion for Mobile Robot Exploration and Navigation, University of Pittsburgh, 1996.

Meng M., Yang X., A Neural Network Approach to Real-Time Trajectory Generation, *Proceedings of the IEEE International Conference on Robotics and Automation*,

Leuven, 1998.

Mirtich B., Canny J., Using Skeletons for Nonholonomic Path Planning among Obstacles, *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, Nice, 1992.

Overmars M., Vleugels J., Kok J., *Motion Planning Using a Kohonen Network*, Utrecht, 1995.

Papadimitriou C., Raghavan P., Sudan M., Tamaki H., Motion Planning on a Graph, *Proceedings of STOC '94*, La Jolla,1994.

Paromtchik I., Garnier P., and Laugier C., Motion Control for Autonomous Maneuvers of a Nonholonomic Vehicle, The Institute of Physical and Chemical Research, Hirosawa, 1998.

Pignon P., Hasegawa T., Laumond J., Basic Algorithms for Space Structuring in Path Planning for Mobile Robots, *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, Nice, 1992.

Pirjanian P., The Notion of Optimality in Behaviour-Based Robotics, *Aalborg University Report*, Aalborg, 1998.

Qin C., Henrich D., Randomized Parallel Motion Planning for Robot Manipulators, Oxford, 1996.

Quinlan S., Khatib O., Towards Real-Time Execution of Motion Tasks, *Stanford University Report*, 1991.

Rajasekaran R., Ramaswami R., Optimal Mesh Algortihms for the Voronoi Diagram of Line Segments and Motion Planning in the Plane, *Proceedings of the 30th Annual Allerton Conference on Communications, Control and Computing*, Monticello, 1992.

Rao V., Rao H., C++ Neural Networks and Fuzzy Logic, First Edition, MIS Press, New York, 1993.

Reif J.,"Complexity of the mover's problem and generalizations", *20th FOCS*, 1979.

Reif J., Sharir M., "Motion Planning in the Presence of Moving Obstacles", *Proceedings of the 26th IEEE Symposium on Foundations of Computer Science*, 1985.

Rosenblatt J., DAMN: A Distributed Architecture For Mobile Navigation, Robotics Institute PhD. Thesis, 1997.

Ruspini E., Saffiotti A., Konolige K., Progress in Research on Autonomous Vehicle Motion Planning, *Industrial Applications of Fuzzy Logic and Intelligent Systems*, IEEE

Press, Piscataway, 1995.

Sankaranarayanan A., Masuda I., A New Algorithm for Robot Curve-Following Amidst Unknown Obstacles, and a Generalization of Maze-Searching, *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, Nice, 1992.

Saffiotti A., Ruspini E., Konolige K., Blending Reactivity and Goal-Directedness in a Fuzzy Controller, *IEEE Neural Nets and Fuzzy Control*, San Francisco, 1993.

Saffiotti A., Ruspini E., Konolige K., "Using Fuzzy Logic For Mobile Robot Control", Chapter 5, International Handbook of Fuzzy Sets, Dubois D., Prade H., amd Zimmerman H. eds. Kluwer Academic Publisher, 1999.

Scheuer A., Fraichard T., Planning Continuous-Curvature Paths for Car-Like Robots, *INRIA Rhones-Alpes Report* , Grenoble, 1995.

Scheuer A., Laugier C., Planning Sub-Optimal and Continuous Curvature Paths for Car-like Robots, *INRIA Rhones-Alpes Report*, 1998.

Schildt H., C++ the Complete Reference, Osborne McGraw-Hill Press, First Edition, New York, 1991.

Schwartz J., Sharir M., "On the piano movers problem: III. Coordinating the motion of

several independent bodies: the special case of circular bodies moving amidst polygonal barriers", Technical Report, Courant Institute, NYU, 1983.

Seidewitz E., Stark M., Reliable Object-Oriented Software:Applying Analysis and Design, SIGS Books, New York, 1995.

Sharma R., A Probabilistic Framework for Dynamic Motion Planning in Partially Known Environments, *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, Nice 1992.

Shampine L., Numerical Solutions of Ordinary Differential Equations, Chapman & Hall, New York, 1994.

Sleuner N., Tschihold-Gurman N., Exact Cell Decomposition of Arrangements used for Path Planning in Robotics, *Swiss Federal Institute of Technology Report*, 1999.

Smirnov Y., Koenig S., Veloso M., Simmons R., Efficient Goal-Directed Exploration, Carnegie Mellon, Pittsburgh, 1996.

Stentz A., Optimal and Efficient Path Planning for Partially-Known Environments, *Proceedings of the IEEE International Conference on Robotics and Automation*, IEEE Press, 1994.

Takeda H., Latombe J., Sensory Uncertainty Field for Mobile Robot Navigation, *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, Nice, 1992.

Sehad S., Touzet C., Self-organizing map for reinforcement learning: Obstacle avoidance for Khepera, *IEEE Proceedings From Perception to Action*, IEEE Press, Lausanne, 1994.

Thrun S., Learning Maps for Indoor Mobile Robot Navigation, *Artificial Intelligence Journal*, 99(1), 1998, 21-71.

Van der Stappen A., Overmars M., de Berg M., Vleugels J., Motion Planning in Environments with Low Obstacle Density, *Technical Report UU-CS-1995-33*, Utrecht, 1995.

Walsh G., Tilbury D., Sastry S., Murray R., Laumond J.P., Stabilization of Trajectories for Systems with Nonholonomic Constraints, *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, Nice, 1992.

Yang X, Neural Network Approaches to Real-time Motion Planning and Control of Robotics Systems, PHd. Thesis, University of Alberta, Edmonton, 1999.

Yang X., Meng M., An Efficient Neural Network Model for Path Planning of Car-Like

Robots in Dynamic Environments, *Proceedings of the 1999 IEEE Canadian Conference on Electrical and Computer Engineering, Edmonton*, 1999.

Zhao Y., BeMent S., Kinematics, Dynamics and Control of Wheeled Mobile Robots, *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, Nice, 1992.

Zimmer U., Puttkamer E., Comparing World-Modelling Strategies for Autonomous Mobile Robots, *Proccedings of IWK'94*, Ilmenau, 1994.

Zomaya A., Morris A., Direct Neuro Control of Robot Manipulators, *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, Nice, 1992.